Università di Pisa

Dipartimento di Informatica

Technical Report : TR-99-21

Learning conceptual descriptions of categories

Giuseppe Attardi, Maria Simi, Filippo Tanganelli, Alessandro Tommasi

November 30, 1999

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

Learning conceptual descriptions of categories

Giuseppe Attardi, Maria Simi, Filippo Tanganelli, Alessandro Tommasi

November 30, 1999

Department of Computer Science – University of Pisa
Corso Italia 40 – I-56125 – PISA – Italy
{attardi, simi, tanganel, tommasi}@di.unipi.it

Abstract

In this work we propose a model to learn conceptual descriptions of categories from precategorized texts. The model is general and parametric, and it captures most of the statistical approaches to classification as well as allowing the definition of more symbolic learning schemes. The algorithm scheme has been instantiated into three different algorithms, which have been implemented and tested on a collection of documents obtained from the Web. As a possible application of the descriptions obtained, classification was done on a test set. Results are somewhat surprising, and stand in contrast with most experiments done in literature, possibly giving hints about a different research direction.

1 Introduction

1.1 Motivation

The use of statistical analysis often allows the treatment of phenomena whose complexity goes beyond our modeling capabilities. As an example, the theory of chaotic systems shows how some complex phenomenon, such as the meteorological condition, is not describable by causal theories; all that can be done is to estimate a probability distribution by observation of the *effects*. The statistical approach is fast and, in many cases, effective.

The intrinsic complexity of natural language has pushed, after the initial enthusiasm for symbolic analysis, towards the study of statistical methods for its analysis. Even though nobody ever "proved" that natural language is chaotic, in practice the most effective algorithms for learning from textual documents (amongst which the ones for automatic categorization) are statistical. While it is true that the ratio between the resources necessary for such an approach and its effectiveness will always justify their study, considering statistical approaches as the only viable approach is quite drastic. If it were not possible to describe language analytically, which is the mean by which we communicate, we would not have any chance to make any formal reasoning about it.

If we think that a different approach to study natural language is possible, however, on the other side we must be aware that it is one of the hardest tasks. Because there is a strict relationship between language and thoughts (we often think in terms of language), understanding

[0]This work has been partially supported by **A.S.I.** (Agenzia Spaziale Italiana) under project "Integration and access to heterogeneous databases" and by **MURST** under project "Intelligent Agents: Interaction and Knowledge Acquisition".

language would mean represent nearly all concepts we can think of. This seems like a very ambitious goal, which we cannot ever expect to accomplish. We strongly believe, however, that working in that direction could give us an impressive amount of intermediate and useful improvements.

In this work we deal with the problem of extracting conceptual descriptions of categories from precategorized documents. We plan to give, for each category, a set of concepts, as described by textual features, which we will interpret as concepts related to the category meaning.

In the rest of this section, we give a panoramic view over the most common issues in learning from categorized texts, identifying the peculiarities of different approaches.

In the second section, we describe two research projects that, although carried out in different contexts, gave us ideas on how to approach the problem.

We describe the model formally in the third section, marking out the many parameters of a general scheme which needs to be grounded further in order to be applicable.

In the fourth section we give three different instantiations of the model, choosing the parameters in order to test the system with three different types of features: single terms, noun phrases and lists of noun phrases.

In the fifth and sixth sections we describe further processing of the learned category profiles. Additional techniques for boosting the training phase are also defined.

Classification using the profiles is studied in section seven. Some of the classical techniques are described, and some original ones are defined.

In section eight, we describe the implementation of a prototype system. We briefly describe some auxiliary modules which helped us in the development as well.

In section nine we present the results, by making a direct analysis of the profiles, and by taking classification ability into account. For the latter, standard measures as well as subjective interpretations are deployed.

The tenth and final section is reserved to conclusions and evaluation of the work.

1.2 Background work and issues

Scientific researchers have recently multiplied their efforts in studies about Information Retrieval, practical applications of which are becoming more and more necessary to end users, and to the scientific community itself, in order to fetch the increasingly available information efficiently.

In the past few years, not only have new documents been produced directly in digital form, thus being suitable for automatic indexing, but also many of the older documents have been ported from their "physical" medium to the digital one. At the same time, the availability of more powerful computers at lower prices gave everybody a chance to produce more documents in digital form.

While earlier studies in Information Retrieval originated from an internal necessity to organize libraries, the increasing interconnection amongst computers pushed the efforts further, in order to provide a service not only to trained professionals over private collections, but also to less experienced user; looking for information on some distributed collection of documents.

There are three different factors that have boosted interest on Information Retrieval:

- old documents conversion to digital form: not only structured data is hence available for computers to treat, but also unstructured, text based one;

- new documents, no matter what their target is, are produced directly in digital form: a wider and more heterogeneous collection of information is then available, thus making a retrieval system of interest for a broader category of users;

- interconnection makes the information publicly available: this causes in some case the *necessity* to use that information. The ability to fetch strategic information from a public network such as Internet gives a consistent advantage over concurrent companies; to protect themselves, companies have to make sure their technological level is up-to-date with present opportunities.

Not unrelated to Information Retrieval is Automatic Categorization, the task of assigning a document one or more categories out of a predefined set, guessing at the document's main topic. If Information Retrieval can be seen as retrieving documents from keywords, Automatic Categorization could be seen as the inverse process of assigning a document one or more keywords (see e. g. [22; 6]).

Organizing documents into a catalog brings significant benefits to the information search process. While with Information Retrieval one can retrieve documents about one specific topic, a catalog allows narrower (and hence more to the point) searches, as well as a more "relaxed" information browsing, during which the user can develop a clearer idea of her needs. This applies especially to the World Wide Web domain, where most users would like to fetch the available documents (or sites) depending on a *category* of interest. When a user is interested in David Lewis's new article about phrasal clustering, a direct search can be exhaustive, but if the user wants "games for Linux", a search would probably bring up documents about games in which the user has no interest at all. Browsing a catalog could take the user to her area of interest with a few clicks.

The ability to classify documents would be useful not only in an environment in which a user browses "unknown" information, but also in the case that the user wants to *delegate* to the system the task to organize natural language based documents, such as word-processor documents, e-mail and the like. Such a form of interaction between the machine and the user would move away from the latter the effort of organizing the work, and thus effectively hiding the file-system to the user. Ideally, documents would be clustered in a way that would match a user's criterion of association, making specific searches fast and effective.

Automatic Categorization is then a highly desirable task to be automized, given the impossibility, for humans, to deal with collections typically very large.

Automatic Categorization involves knowledge at two levels: knowledge of the meaning of the category, to understand what pertains to it, and knowledge of the meaning of a document, to decide whether or not it matches a category [3]. An effective and robust representation for *meanings* is hence desirable, an achievement of which would grant the chance to build a highly trustworthy and effective automatic classifier.

Studies on Automatic Categorization have typically exploited statistical and probabilistic peculiarities of syntactic *features* of documents, such as the occurrence of terms or lists of terms. The meaning of a document is then often represented by a vector of such features, which are weighted according to a measure which is intended to be an estimation of its relevance relatively to the document.

While the meaning of a document is "approximated" directly by the examination of its content, the meaning of a category is often induced by the examination of a set of documents belonging to it, according to one of several machine learning approaches. Techniques for learning from a set of examples are widely studied, as they have several practical applications and, as machines get more and more powerful, and able to treat a larger number of examples quickly, they are performed at lower cost.

In a typical learning method a set of feature is defined over the domain, in order to identify characteristics upon which to base the description of a predictive rule. Such features can be low-level (and therefore with a weak abstraction capacity) such as the pixel value in an image, or higher-level (thus "interpreting" domain properties) such as, still for images, shape properties.

A set of features must therefore be defined over text documents for these methods to be applicable. By choosing syntactic features such as the terms used and their frequency in the documents, learning methods make the assumption that the meaning of a document is not independent from the syntactic form of its content. While this is reasonable in many cases, sophisticated texts containing communication devices like metaphors can mislead the system. From a different viewpoint, one could argue that deeper text understanding is infeasible with a syntactic approach, since while it would make sense to ask about the topic covered by a document, it would not to ask about the ideas expressed, if only syntactic features are to be used.

By representing documents with vectors such as TF-IDF weighted terms this assumption is obvious, and the learning method applied will inevitably follow the same rule. The most successful learning algorithms for text categorization use single terms as features, that is, they represent the meaning of a document or of a category by a vector of weighted terms (see e.g. [11; 18; 13; 16]). This stands somewhat in contrast with the intuition that a term is not very representative, and that some sort of natural language processing (although shallow) would be of great benefit to understand the meaning of a text. Experiments of systems using more sophisticated structures, such as *n-grams* (lists of n terms) or noun phrases have proven to perform quite disappointingly, especially considering the extra effort required by an analysis that goes beyond simple terms. Practical results have shown that performance is, in these cases, well underneath typical results obtained with the use of single terms.

Although with statistical methods the trade-off between practical results and computational effort is very convenient, there are several fields of application where an even small improvement would be greatly desirable, no matter what the cost of it is. If, for example, further language understanding (of any kind) is required e. g. to provide the user with some higher level form of interaction with her computer, it seems obvious that a purely probabilistic method would not be of help, since it seems like their performance is limited to a maximal accuracy of about 75%. For as much as these methods have proven effective and (most important) very stable, we believe that further study to solve the problem in a more analytic way could open a whole new perspective, even if only partial results are achieved as intermediate steps.

We propose here a model for the construction of description of categories from categorized texts. Such descriptions are intended to represent concepts related to the category in a semantic way, thus describing the category in an "analogic" way, listing the concepts that either "belong" or are at least strictly connected to the category. An example of a good description of this kind could be the following list of concepts, associated to the category "Music": "note", "song", "musical instrument", "rock band", "album chart", "melody", "string (of guitar, piano, violin etc.)" and so forth. Descriptions should obey to an "economy" principle, that is, they should only identify what *distinguishes* the category from the others in the learning environment, in order to focus on peculiarities of each category and hence obtain a more specific and to the point representation for it.

Such descriptions are not, per se, solving the categorization problem, nor allow the building over them of a higher level form of interaction with computers, but they represent an intermediary step which is intended to show how a meaning representation other than flat collections of weighted terms is possible, and how some form of natural language processing, combined with machine learning algorithms, can be effective at identifying relevant concepts in documents. Obtaining these descriptions poses the basis for further study on systems for learning from text which perform operation in a "user-like" fashion instead of classifying according to some "cryptic" classification rule.

The descriptions can be used to perform some prototypical kind of categorization, exploring the opportunity to use the identified concepts as indexing features for documents. Another use could be the one of integrating linguistic tools, enriching them with relationships between concepts which are specific to the domain or which emerged as the language evolved.

1.3 Terms-based classifiers

The model we propose describes a learning algorithm over textual documents. Many classification algorithms are defined over the same domain, and, since the model we propose tries to generalize them, we describe them and identify their basic assumptions.

The most successful classifiers use learning algorithms over the terms contained in the text rather than on more sophisticated textual structures. Although these algorithms could be applied to any syntactical structure, they are most commonly used in association with the "bag of words" assumption, that is, that a text is classifiable by knowing what words occur in it, with no clue about e. g. their position in the text.

As far as a system using the "bag of words" assumption is concerned, the text:

Mary had a little lamb

is perceived just as (and then has exactly the same meaning of):

A little lamb had Mary

or as:

Little had Mary a lamb.

One could think that the longer the text is, the more significantly its meaning changes by changing the words' order, thus making the bag of words assumption stronger. As a matter of fact, as the text becomes longer, an important information becomes available and significant to the system: the *frequency* of words. By the bag of words representation, it is still possible to recognize that a word occurs very frequently, which makes it reasonable to assume that it is more important to the meaning of the document with respect to a word that has a very low occurrence frequency. By representing this section like a bag of words it would not be surprising to see that (other than extremely frequent and "meaningless" words such as "the") the words "document", "category", "classification" are amongst the most frequent.

The bag of words assumption is the easiest way to solve the problem of choosing the features to represent the meaning with. There is almost no processing at all to choose such features, and even irrelevant terms are put in the "bag" representing the meaning of a text. In the aforementioned representation of this section, for example, the word "lamb" would appear, since it occurs, but it is irrelevant (if not misleading) to the meaning of it.

Little effort is therefore done to *choose* the features to use for the learning algorithm. The features obtained will subsequently be evaluated by assigning them a *weight* which measures their importance relatively to the document or to the category. We do not go into details on how this measure is evaluated: it is only relevant, here, that all these measures depend on the number of occurrences of features.

The most popular learning algorithms differ for the choice of the measuring function. A weighting technique for the importance of terms in representing the meaning of a document is known as TF-IDF, and it is explained in detail later in section 5.1. Whether or not a document is represented by TF-IDF weighted terms, these terms are combined together in a vector of weighted terms representing the meaning of the category by algorithms such as Rocchio's, Widrow-Hoff or Naïve Bayes.

A different approach is the one used in the k Nearest Neighbors classifier (kNN). kNN does not perform any kind of explicit extraction from the documents used as training set, but identifies the learning process with the ability to store examples. This method stores all examples along with their classification, and when the classification of a new document is requested, it performs it by looking at the classification of the k examples most similar to it. The similarity between documents is usually performed by comparing the two vectors representing the documents. Each vector is typically formed of weights assigned to terms by a measure such as TF-IDF: in this case kNN still uses the bag of words representation for documents. Because no explicit extraction of information is performed, kNN does not describe a category in any way (except by the set of examples belonging to it), but it has been proven to perform remarkably well in classification [12; 16; 18; 34].

The bag of words representation of documents (and therefore of categories) is not appropriate to explicitly represent the meaning of a document or of a category. The profiles generated, if any, are long vectors of pairs word-weight, and although they can provide a classification rule, they cannot be used for other purposes. Simple independent terms are not enough to communicate a meaning: one cannot assume that each word represents a concept. Terms can form concepts when aggregated in phrases, but when taken in isolation they suffer from problems such as *polysemy* (multiple meanings for one word), *synonymy* (multiple words for one meaning), or undesirable generality. In most cases, the generality of the meaning of abstract nouns makes it impossible to communicate an actual unambiguous concept by one of them. If we independently consider the words: "stock", "group", "share" and "match", we might have trouble assigning a precise meaning to them. If, instead, we make their meaning more specific by adding specifiers, we might have: "stock and barrel", "music group", "share in a company", "lit match", whose meaning is easier to categorize.

1.4 Classifiers based on compositions of terms

While the bag of words representation seems to be the simplest starting point for indexing documents (for automatic categorization or information retrieval), one could think that further efforts in indexing more complex textual structures would significantly improve the performance of a classifier. Much research has been done on the use of structures such as lists of terms (n-grams) to index documents, in the belief that the effort would have resulted in an improved efficiency.

There are two main alternatives to consider structures more sophisticated than terms: one can perform statistical analysis, indexing the structures that co-occur more often or more significantly, or one can do some natural language processing, identifying structures that have some syntactic form. Example of the former are the n-grams, while an example of the latter are noun phrases.

The pure statistical method assumes that the more features one can index, the better the meaning can be represented, even if still no explicit meaning can be associated to each feature. The methods that use natural language processing, conversely, identify syntactic structures with a clear meaning, and use these to index documents and represent the meaning of a category.

While, for example, with the first approach a good feature to index a document could be a list of two adjectives, the latter would require the feature to represent a concept, and be e. g. a noun phrase.

Despite the many efforts, however, such researches have shown that the use of structures more sophisticated than simple terms does not result in an improved performance. Quite disappointingly, many times it has proven to drop down accuracy significantly. The reasons for this unexpected result are still to be fully understood, but we can make some objective observations.

For the statistical based approaches, we must consider that while each term has a high number of occurrences, and then statistical data about it are significant, in most cases textual structures are rare. This determines incorrect estimations and can make classification less effective.

Approaches based on natural language processing techniques suffer from the lack of such techniques in representing language effectively. The many assumptions made when processing natural language are (apparently) too many or too strong, and as the meaning of a document cannot be fully understood or effectively represented, its classification is likely to be wrong.

Examples of work exploiting more sophisticated features instead of single terms are e. g. [7; 14].

There are, however, interesting aspects about the use of natural language which makes it worth further study. First of all, while statistical analysis is not likely to be improved, many hope that a better language understanding could be achieved as more linguistic tools become available. Also, for the current purpose of classification, a full text understanding is not required. Concentrating on weaker assumptions and trying to simply understand the *topics* of a text, for instance, would allow us a good classification performance even if we do not understand relationships amongst these topics.

2 Relevant work

Previous work in linguistic analysis and classification techniques can be exploited directly in this work, as the application domain is the same. To improve the results, though, a learning scheme is described, approaching the problem from a different viewpoint. Ideas to develop such an algorithm were taken from previous work on different topics: a description of such work helps understanding the system.

2.1 Luc Steels: perceptually grounded meaning creation

Luc Steel's[1] work is an investigation on the origins of the language, which he simulates by a population of agents which interact by communicating. He has done many experimentations of great significance (some of them publicly exposed at museums or in progress on the Web), based on either computer simulation or physical robots, and has achieved interesting results.

The part of his work we focus on is the one where he states that an agent creates a concept to discriminate among objects [28; 29]. He describes a process by which an agent perceives the world, builds concepts from these perceptions, and shares them with other agents that have undergone different experiences.

Steels defines an environment which contains objects and agents. Moving in this environment, the agents observe objects and build a conceptual description of them based on their

[1] He works at the Artificial Intelligence Laboratory of the University of Brussels and at the Sony Computer Science Laboratory in Paris.

perceptions. An agent, when meeting another one, tries to communicate with him about an object, eventually sharing not only meaning but also words.

The learning of conceptual descriptions has an auxiliary relevance in Steels' work, since its purpose is simply to create a perceptually grounded meaning to share with other agents in the form of language. For this reason the learning scheme is simple, yet it gives some interesting ideas.

2.1.1 Agents, sensory channels and features

An agent is autonomous and supplied with sensors; it can move in its environment and perceive objects.

The sensors are formally functions from the set of all objects in the environment ($\mathcal{O}$) to the real interval $[0, 1]$. As suggested by Steels, low level routines could be implemented to obtain higher level perceptions combining the readings on the sensors (and thus creating a sort of virtual sensory channel).

Steels identifies *features* by "discretizing" the values obtained from the sensory channel. Dividing the interval in two subintervals labeled v_0 and v_1, Steels identifies two features as pairs attribute/value, where the attribute is the sensory channel and the value is either v_0 or v_1 depending on where the reading on the channel lies.

An agent begins to observe an object with the lowest possible resolution, considering a sensor and looking whether the reading falls in the v_0 or v_1 interval. As the agent starts getting confused with different objects that share perceptions, it refines the reading, by subdividing the interval further. The agent constructs then binary trees to represent the discretization of the reading of the sensors.

Each subdivision of the interval corresponds to an attribute which, paired with the indication (v_0 or v_1) of where the reading lies, forms a feature. The name of an attribute has the following form: `sc-`n`(-2)`$^+$, where "`sc`" stands for sensory channel, n is the number of the sensor, followed by a number of "`-2`" equal to the number of refinements over the channel. For example, considering the fifth sensory channel, `sc-5-2` is the attribute that corresponds to the first subdivision of the reading interval of that channel in two parts. The two features based on this attribute are indicated by `(sc-5-2` v_0`)` and `(sc-5-2` v_1`)`. Consistently, the attribute `sc-5-2-2-2` corresponds to the same sensors, whose reading interval has been divided into subintervals three times (the binary tree has depth 3). An example of a tree is shown in figure 1.

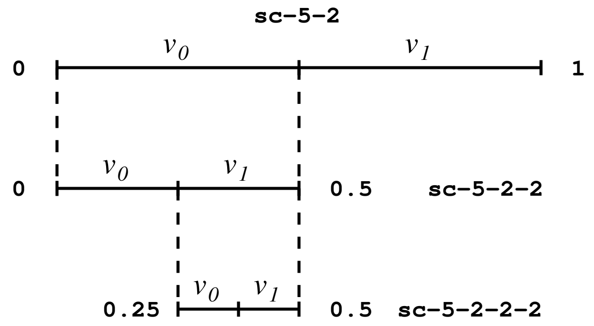

Figure 1: **an example of refinements in the reading of the fifth sensor channel. The initial reading corresponds to the value of the attribute `sc-5-2`, while successive refinements cause the definition of the attributes `sc-5-2-2` and `sc-5-2-2-2`. v_0 is always the left interval, while v_1 is the right one.**

2.1.2 Discrimination games

The agents move in the environment observing the objects they meet and engaging in what Steels calls *discrimination games.* An agent considers a subset $\mathcal{O}'$ of $\mathcal{O}$. The agent has perceptions for each object in $\mathcal{O}'$. To begin the "game" an agent selects an objects in $\mathcal{O}'$ as the *topic*, and considers all other objects in $\mathcal{O}'$ as the *context.* The game is to find a set of *distinctive* features to discriminate the topic from the context. Initially, the agents has no attributes, but in case of necessity new attributes are formed by considering another sensory channel or by refining one further.

Formally, two features $f = (a\ v)$ and $f' = (a'\ v')$ are *distinctive* if $a = a' \wedge v \neq v'$.

A set of features F is distinctive of an object o_t with respect to a context C if $\forall f \in F. f = (a\ v) \wedge (\forall o \in C.(\nexists f' = (a'\ v') \in F_o.a = a') \vee \exists f' \in F_o.f$ and f' are distinctive), where F_o is the set of features for an object o. The definition says that the set of features for the topic and for the context must contain features based on different attributes, or at least one feature for each object in the context must differ for the topic.

When a set of distinctive features is found for the topic, the game ends in success. When such a set cannot be found the game fails, and the agent refines one of its perceptions to try and discriminate the topic from the context. If, for example, the agent was perceiving the features $\{$(`sc-5-2` v_0), (`sc-6-2` v_1)$\}$, it could read another channel, obtaining for example (`sc-3-2` v_1), or refine the first perception introducing (`sc-5-2-2` v_0).

By reading the sensory channels with low resolution the agent keeps the features general. When such generality is confusing the agent, which is not able to discriminate a topic from the context anymore, more specific features are created. Descriptions move away from a general horizon to a more specific one according to the needs of the agent.

As shown by experimental results performed by Steels, the agent eventually converges towards a set of features sufficient to discriminate every possible topic from every possible context, still leaving the world open for the arrival of new objects.

2.2 Considerations

The learning scheme has the auxiliary goal to give the agents something to communicate to each other. In Steels' environment, agents also engage with other agents in *language games*, with which, by communicating distinctive feature sets, they form and share a language.

The features that an agent will develop to describe the world are not known beforehand. In a way, such features represent abstraction from the low-level perceptions, and give an agent a way to *describe* objects. Every feature can be considered like a concept that the agent has developed in order to be able to understand the situations that are surrounding him, in the form of topics and contexts.

Agents' first attempt at abstracting the information read from the sensors is represented by the first division of the reading interval in two subintervals. As this is a rough and heavy discretization, it might likely not be enough, and the agent will have to refine its perception, by subdividing more, therefore giving a finer discretization of the reading interval. In this case, the first division has proven to be somehow *ambiguous*, because it could not help telling apart an object from a context.

2.3 The analogy

Our goal is to build an agent that learns conceptual descriptions of categories in a way similar to the one in which Steels' agents learn in their world. Sensory channels are, in our case, the text of the training set examples, while the objects in Steels' environment correspond to the categories.

By observing the same object from different viewpoints one has different perceptions: each example is a different perception of the same object, the category to which it belongs. The features of an object, the profile of a category in our analogy, are obtained with respect to the *context* of other objects/categories. The descriptions are therefore able to discriminate a category from another.

The way that such ability will be achieved is by keeping features as general as possible until they do not allow to tell apart two categories: in that case some form of refinement like in Steels' discrimination games will be necessary.

2.4 ART: Adaptive Resonance Theory

Adaptive resonance theory was introduced by Stephen Grossberg in 1976, as a study about models of neural networks. Reported here is the description given by Heins and Tauritz [9].

Adaptive resonance theory was developed to define a computational model similar to the human brain. One of the results of such study has been the definition of an algorithm, called ART algorithm, to cluster inputs into groups of inputs similar to each other. Clustering algorithms represent a very important form of unsupervised learning, and have been deeply studied. Amongst them, the ART algorithm is widely used, and variants of it (e. g. [4]) exist to extend or specialize its behavior.

2.4.1 The main idea

ART is an incremental algorithm. It builds a prototype representing each cluster: every new example is compared to the profile and assigned to the relative cluster if close enough. The profile is then modified to be more like the example. If none of the prototypes are close enough to the example, then a new prototype (and therefore a new cluster) is created. The prototype will be equal to the example, which will also be the only member of the new cluster.

This scheme intuitively follows what happens whenever we experience a member of a new class: it will become the only representant of such class, and all other objects will be compared to it to determine whether they belong to the class. When more examples of the same class are seen, we generalize from the individuals and form a more abstract description.

2.4.2 The formal definition

The algorithm ART-1, of which we give the formal definition here, clusters binary vectors of length n by constructing a set of prototypes. Let ρ and β, whose meaning will be discussed later, be values in the real interval $(0, 1]$. P is the set of prototypes.

1. $P \leftarrow \emptyset$;

2. if there are no more input vectors then stop, otherwise get an input vector I. Let P_I, the set of prototypes that represent the possible clusters to which I belongs, be initialized as P;

3. find i that maximizes $\frac{\|p_i \cap I\|}{\beta + \|p_i\|}$, where $x \cap y$ is the bitwise or of the binary vectors x and y, $\| \, . \, \|$ is the "magnitude" of a vector, that is, the number of 1's it contains and p_i the elements of P_I;

4. check whether $\frac{\|p_i \cap I\|}{\beta + \|p_i\|} < \frac{\|I\|}{\beta + n}$: if so, create a new prototype equal to I and insert it into P, go back to step 2;
5. if $\frac{\|p_i \cap I\|}{\|I\|} < \rho$ then $P_I \leftarrow P_I \setminus \{p_i\}$; if P_I is empty go to step 2, otherwise go to step 3;
6. $P \leftarrow P \setminus \{p_i\}$; $p_i \leftarrow p_i \cap I$; $P \leftarrow P \cup \{p_i\}$; go to step 2.

Each prototype represents a cluster. To determine whether an input belongs to a cluster or to none, the algorithm finds in P_I the prototype that maximizes $\frac{\|p_i \cap I\|}{\beta + \|p_i\|}$, that is, the one that share more 1's with the input vector. The denominator makes sure that, in case of ties, the prototype with fewer 1's win. $\beta > 0$ ensures that the denominator is non-zero.

The fourth step ensures that the prototype has not been selected just because the input vector has plenty of 1's: in that case, it is likely that it will be close to any prototype. If this is the case, the vector is assigned to a new cluster, of which it becomes the prototype.

The fifth step controls that enough 1's are shared by the input vector and the selected prototype: if too few 1's are shared, the prototype would become too general. ρ makes sure that no prototype becomes too general. If the selected prototype does not verify this condition, then the algorithm looks for another one.

Once a prototype is found (and hence a cluster is assigned), it is generalized by doing the bitwise *and* with the input vector. This way, at any time the prototype represents all the 1's shared by all members of the cluster: these 1's are the common characteristics of the members of the cluster.

2.4.3 Considerations

The algorithm constructs prototypes that represent all the features common to members of that cluster, but that can also be found in members of other clusters. The role of the parameter ρ is to *limit the generality of the prototypes*. When a prototype is updated to be more similar to the new input vector, ρ controls that this step, that abstracts features from the input vector, is not excessive, producing an overly general prototype.

The value of ρ gives a mean to control the granularity of the clusters: when ρ is close to its maximum value, 1, it will force the formation of cluster whose prototypes are very similar to the vectors contained in the cluster. The granularity will therefore be very fine, since there will be many cluster each of one containing few vectors very similar to each other.

As the value of ρ is lowered down towards 0, the system will start behaving differently. Fewer clusters will be created, and they will be broader, containing more vectors which will be less similar to each other.

Because ρ smoothly changes the behavior of the algorithm from the extreme of one loose cluster to the one of one cluster for each input vector, it is called the *vigilance parameter*.

2.4.4 The analogy

The ART algorithm suggests us many useful things for our purpose. The vigilance parameter effectively controls the generality of the prototypes, and can be varied to obtain different behaviors. The unsupervised learning scheme allows the identification of relevant features without introducing external information.

An algorithm such as ART could be applied to the documents belonging to the same category in order to "cluster" them around the features they share. We can imagine documents in a category as naturally organized in groups, each representing a feature which those documents share. In a category labeled as “Tennis”, for instance, we could organize documents around the major tennis tournaments, or around the tennis players.

We have then an analogy in which the clusters correspond to features, or set of features. Because, however, the features are not necessarily a partition of documents, we cannot apply the algorithm directly. The features that the document will share in a category will not actually be clusters, so the learning scheme will have to be changed.

A document will possibly be a member of more than one “cluster”, that is, more than a feature might be identified inside of it.

The vigilance parameter will not be a static value chosen before the run, but will be a variable associated with each feature, in order to control its generality individually.

3 The learning model

3.1 The scheme

The ideas described in the previous section can be combined together in a new learning scheme. The intuition is as follows.

The texts in the training set are “perceptions” of the object we want to learn a description: the category. Such descriptions will contain the characteristics that the documents belonging to that category have *as opposed to the others.* Learning the description of a category will therefore be subject to the *context* in which the learning process is performed. If we show an apprentice only members of the categories “Ferrari” and “Harley Davidson”, he might as well think that “Ferrari” is what has 4 wheels, while “Harley Davidson” is what has only 2. But as we show the apprentice “Ferrari”, “Seat” and “Harley Davidson”, he will realize that saying that Ferraris have 4 wheels is not enough: he also have to say e. g. that they consume more than 15 liters of gas per 100 kilometers, to discriminate them from Seats.

The more compact the description will be, the better. If some sort of minimum description length principle can be seen here, it is true that over two equivalent descriptions, we would rather adopt the shortest one.

ART will give us a mean to extract the features common to the documents of a category. Consider the category labelled as “Sport”: if a document talking about soccer and a document talking about tennis do not share much, if anything at all, much more would be shared by two documents about the same sport. Documents will then have, in groups, common features, upon which the algorithm should build the descriptions.

As opposed to ART, which deals with disjoint clusters, a document will have to be assigned not only to the closest group of documents (representing a feature), but to all of the groups it is “close enough”. The notion of “close enough” will depend from a value associated with the feature itself, and this will change during the learning process as more generality of specificity of the feature is needed.

Because documents share more than one feature, we can expect that, if a new document fits in no group, it might still share some good feature with other documents, which had not been identified before. For this reason, when a new group of documents is created (i. e. a new feature), all the past documents are checked against the new group, to see if they belong to this one as well.

The variable value associated to each feature in order to control its generality will vary when the feature shows to be too general. As seen on Steels’ work, features should tell one category apart from another: when a feature is too general it cannot guarantee that property, and should be made more specific. The "perception" of the text will then be refined, resolving what stands as an *ambiguity*.

For each category, we should then obtain features that have been found common to documents belonging to the same category, but that, at the same time, are not as general as being ambiguously common to documents belonging to another category. ART's mechanism of controlling the granularity will allow us to group the documents of a category around proper features, while Steels' model to refine ambiguous features will tell us which features should be refined, staying as much as possible on the general side, moving to more specificity only when needed.

3.2 Formal description of the model

The model we define will be general, and different instantiations will cause different behaviors of the system. Many parameters will be left unspecified, since their specification corresponds to assumptions that do not belong here, but a description of their meaning as well as suggestion for legal choices will be given.

As a first step, we introduce the terminology that will be used throughout the rest of this report.

feature

In this context, a *feature* is a characteristic of a text. It can be anything from the language the text is written in, its length, the occurrence of a term or of a sequence of terms. Features will be used to represent *concepts*, assuming that if a certain feature occurs in the document, then that concept is there expressed.

document

A *document* is a text, and it is described in terms of features that appear in it.

profile

A *profile* of a category is a set of features that represent the category. It can directly represent its meaning, or it can just provide a rule to determine whether or not a document belongs to the category. A numerical weight is usually associated to the features in the profile, so as to mark the more important ones as opposed to the least important.

example

An *example* is a classified **feature**, that is, a pair

$$e = \langle F_e, L_C \rangle$$

where F_e is a feature and L_C is the set of categories to which the example belongs. Usually, whole documents are considered as training examples. For our purposes, we will consider each feature in a classified document as an example itself, simplifying the description of the process.

The goal of the learning process is to obtain a profile for each category for which training examples are provided.

The same terminology can be used to describe traditional algorithms like Rocchio's or more experimental ones, based on compositions of terms. The first ones usually have terms as features, while in the second case features could be n-grams, noun phrases and so forth. Features are obtained from the training documents, which are searched for the selected textual structures.

The definition of example given above differs from the one traditionally used in those cases, but it applies nevertheless. Because no data is usually available about the co-occurrence of features, each one is typically considered independent: with this assumption, one could as well treat each feature as an independent example. Treating each feature as an example is a big approximation, since not all features of the specified kind in a text can be good evidences of the category to which the entire text belong. We know that the *document* belongs to a category, not the *feature*. Consider the following text [2]:

Wars, whether they're waged between nations, corporations, or individuals, are launched for myriad reasons– some of them rational.

This text could be classified under a category such as "politics". Of the 18 terms, though, only "wars" and "nations" can make us think of the category.

The most important features for a category will emerge only when a weighting measure will be applied. Since no measure based on statistical properties can get the relationship between the meaning of a category and syntactical structures found in documents belonging to that category[3], we will approach the problem stressing the symbolic learning part. Instead of simply collecting features, in order to weight them with some relevance measure in a successive phase, we will obtain relevant features with a learning algorithm. The features obtained will still be weighted after that phase.

Because we want to keep the model as general as possible, we will define the features by a tree whose leaves represent basic features.

3.3 The feature structure

We will say that the terms are the basic feature upon which to build more sophisticated structures. Different choices for the basic features could however be taken.

We will say that the *depth* of a feature is the depth of the tree structure which represents it. Simple terms will therefore have depth 0, while noun phrases will have depth 1, since they are basically a list of terms. More complicated structures can be used, such as lists of noun phrases, which have depth 2.

Defining the feature structure in such a general way gives us the chance to explore the most effective structure to represent concepts. Features of depth 0, simple terms, as argued before will be inappropriate to represent concepts. Features of higher depths, such as depth 1, are more expressive, and will therefore be more effective at representing precise concepts. In general, the higher the depth is, the larger part of text will be represented, thus making the meaning narrower and more precise. If features of depth 0 are on one extreme, on the other one there are features corresponding to the whole document: their expression of the original meaning of the text will then be perfect, but we could achieve no generalization.

To "map" a document into textual features of the selected depth (the structure of the feature is therefore the very first step towards an instantiation of the model), we will also represent a text by a tree structure. If we use features of depth 1 representing noun phrases, we could represent a text by a tree of depth 2: a root with a branch for each noun phrase in the text. In this case the features will be a subtree of the structure representing the text because a noun phrase is only a substructure of the whole text. We can organize a document into different structures: consider the text[4]:

[2]Excerpt from Vanity Fair of July 1999

[3]There is no proof for that: it is our strong belief, and our axiom.

[4]Excerpt from the GNU Manifesto, Richard Stallman, ©1985, 1993 Free Software Foundation, Inc., as it

The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important.

If we represent the above text as the set of terms which occur in it, its representing structure would be the one shown on figure 2.

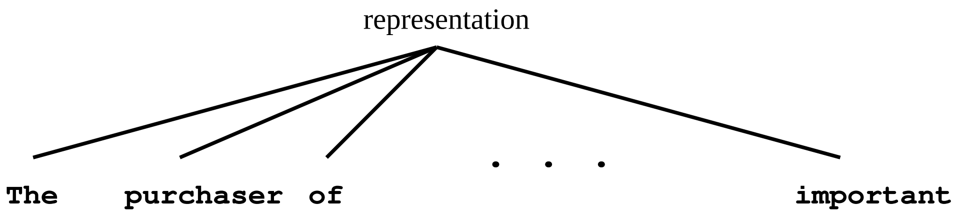

Figure 2: **a representation of depth 1.**

If, instead, we represent the text as the set of the periods which form it, its representing structure will have depth 2, since terms will first be organized into periods, and then the periods will be tied together to form the whole text. An example of such a representation for the same text of the former example is shown on figure 3.

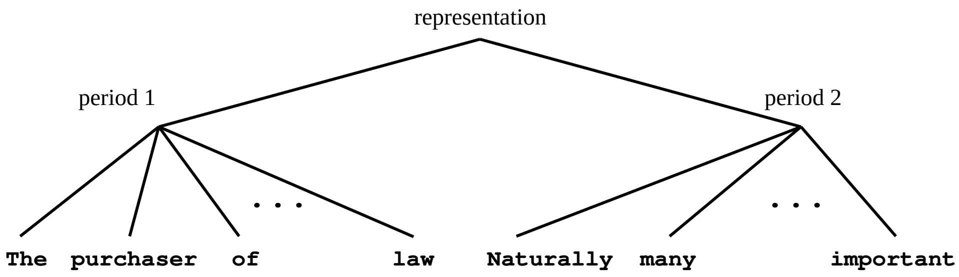

Figure 3: **a representation of depth 2.**

The depth of the tree representing a document does not necessarily have to be greater than the depth chosen for the features. A document and a feature could have the same depth. In this case, the features will have the same structure as the documents, and will therefore represent a sort of *prototype* of documents, rather than an indexing feature. When the features have a lower depth than the documents representation, each document will in fact be treated as a vector of features. Because the profile of a category will also be a vector of features, it will be in the same vector space as the documents, and will represent a prototypical document. If, on the other hand, the features have the same structure as the documents, the profile will be in a super-space, and will not represent a prototypical document anymore. Because in many cases there is no prototypical document effectively representing the meaning of a category, the latter approach seems to bring some advantages.

As an additional advantage, when the features have the same structure as documents, each document will be treated as a feature itself. Because each feature is an example, and any example is assumed to be independent, we are therefore assuming, in this special case, that each document is independent. Since each document *is* independently assigned or not to a category, what once was a simplifying assumption is now simply true. As a consequence, when we choose

appears by typing `CTRL-h CTRL-p` in Emacs.

to have features with the same structure as documents, we know for sure (with the exception of noisy data) that each feature is a good evidence of the category it is assigned to.

On the other side, when we will extract features of depth $k-1$ out of documents of depth k, all information about the co-occurrence of the features in documents will be lost.

3.4 Partial ordering over the features

We define a partial ordering over the feature, representing the higher *generality* of a feature with respect to another.

We call this partial order relationship *subsumption*, and its definition depends on the structure of the features. Subsumption can be defined over syntactical properties of the features, or could be obtained form another source, like a thesaurus. We will denote that F_1 subsumes F_2 by writing:

$$F_1 \sqsupseteq F_2$$

which means that the feature F_1 is more general than F_2: the concept it expresses is more vague. We will also use the inverse relationship by writing: $F_2 \sqsubseteq F_1$.

For example, if we have features that represent the periods in the text, which are sets of terms, and have hence depth 1, the subsumption relationship could be the relation $\subseteq$ over the sets representing the periods. A feature representing the period "it's a nice weather, today" would be more specific than the feature representing the period "it's a nice weather".

The ordering is obviously partial, since there are many cases in which two features cannot be put in any order.

The depth of features will have a role in the definition of the subsumption relationship. It will be impossible to define a non trivial relationship based on pure syntax over features of depth 0. By definition, this features are atomic, and therefore only equality can be tested. All features of depth 0 will therefore be considered not related. One can overcome this situation when linguistic tools such as thesauri are available: if features of depth 0 are terms, from a thesaurus one can extract the information that "mammal" is a more specific term than "animal", and a more general term than "dog". The use of such linguistic resources is still non-trivial, though, for the intrinsic ambiguity of many terms, and will therefore not be exploited in this work. Also, the availability of such resources is still an issue for many languages.

By defining the subsumption relationship over features we also implicitly assume that the concepts connected to the features follow the same relationship. This may not be true in some case, and it is therefore to be considered an assumption that should possibly be addressed more specifically in future works.

3.5 The relation "pertains"

We define a relation of "pertinence": let d be a document and F a feature, we will write:

$$d \rhd F \triangleq \exists F' \in d . F' \sqsubseteq F,$$

and read "d pertains F".

When the features have depth 0, the relation holds when a document contains the term that represents the feature. In the case of deeper features, like for noun phrases, the relation holds when the document contains the noun phrase or a more specific one.

This definition of the relationship is due to the assumption that if a document has a certain topic, then it automatically has (or at least assumes) more general topics as well. For example, if a document talks about a "Bach's concert", then it is assumed that the concept of "concert", more general, is also expressed.

From the definition, we can move that:

$$F_1 \sqsubseteq F_2 \Rightarrow \{d|d \rhd F_1\} \subseteq \{d|d \rhd F_2\},$$

that is, the class of documents pertaining to more specific concepts is a subclass of the one of the documents pertaining to more general concepts. The proof goes as follows, beginning from the transitivity of the relation "$\sqsubseteq$":

1. $F_1 \sqsubseteq F_2 \Rightarrow (\forall F. F \sqsubseteq F_1 \Rightarrow F \sqsubseteq F_2)$
{the consequent of 1 implies the one of 2}
2. $F_1 \sqsubseteq F_2 \Rightarrow \forall d.(\exists F \in d.F \sqsubseteq F_1 \Rightarrow \exists F \in d.F \sqsubseteq F_2)$
{by definition of $\rhd$}
3. $F_1 \sqsubseteq F_2 \Rightarrow \forall d.(d \rhd F_1 \Rightarrow d \rhd F_2)$
{switching to the sets}
4. $F_1 \sqsubseteq F_2 \Rightarrow \{d|d \rhd F_1\} \subseteq \{d|d \rhd F_2\}$

which is our thesis.

3.6 Ambiguity of a feature

The use of natural language terms in information retrieval is subject to the problem of *ambiguity*. Because a term can have multiple meanings (*polysemy*), the use of terms as search keywords can cause the retrieval of irrelevant documents, in which the term has not the meaning it had in the intention of the user. A similar problem can arise when very general terms are used (for example, most of the abstract terms such as "phase", "structure", "mechanism"). Although not strictly polysemic, a very general term does not allow an appropriate discrimination between two (or more) categories.

Using deeper structures, although the polysemy problem practically disappears, excessive generality is still an issue. If we think of a very general phrase, such as "transient phase", we realize that such concept would still be too general to be classified.

To avoid excessive generalization, which we will call *ambiguity* of features, we define a predicate with the purpose of telling us whether a feature is ambiguous or not. While an actual definition of the predicate is left at instantiation time, in order to give another "knob" to turn and experiment with, we can give generic hints about how and what to consider ambiguous.

The definition of ambiguity should not only depend on the feature itself. It is hard to tell whether a feature is ambiguous without looking at the world it resides in. When we say that "transient phase" is an ambiguous concept, we implicitly think that the phrase does not "ring a bell", i. e. , nothing of what we know comes to mind. For this reason, the definition of ambiguity should depend on other features as well.

More specifically, one can say that a feature in the profile of the category C_1 is ambiguous when a more specific feature appears in the profile of another category C_2. This situation means that no document pertaining to the feature of C_2 could not pertain to the one in C_1. The feature introduces a relationship between the two categories which is, in general, undesirable. This assumes that all categories are disjoint.

When categories are not disjoint, we will release this assumption.

The ambiguity of a feature could be the constantly false predicate. In this case, the ambiguity check is omitted, and the system would be allowed to describe a category with any feature. This is, for instance, what we would do if we wanted the system to perform like a term-based classifier, constructing all descriptions of categories from the same feature set. Because nothing can be done with ambiguous terms, this is the best solution when using features of depth 0.

3.7 The algorithm

We will now define a class of algorithms for the creation of category profiles.

The algorithm will be based on two main procedures:

- the *generalization* of example and features, in order to extract information from them;
- the *specialization* of ambiguous features.

During the learning process, a feature in a category profile will be moved towards the more general or more specific horizon until it is able to put together examples of the same category without being ambiguous.

For procedural clarity, let us define a new object: the "profile feature". A profile feature is a pair:

$$\langle F, \theta \rangle$$

where F is a feature and θ is a real value. Profile features are nothing but features with a value attached, and they will form the profile of a category. θ will be used during the training process.

In the following, we will use the notation:

- P_C for the profile (set of profile features) of the category C;
- E_C for the set of examples belonging to C examined so far. We assume that examples are examined in sequence, E_C storing past examples.

3.7.1 The generalization of features

This operation generalizes examples (features of the training documents) belonging to a category in order to find commonalities. Let $F^P = \langle F, \theta \rangle$ be a profile feature in the profile P_C. Let $e = \langle F_e, \{C\} \rangle$ be an example. The generalization of F^P and F_e tries to find a feature F' more general than both F and F_e. Generalization of a profile-feature is limited, not unlike what happens in the ART algorithm, by the threshold θ.

The specificity of a feature F is measured by a function that we will call $\mathcal{G}(F)$. The function $\mathcal{G}$ must grow monotonically as specificity increases:

$$F_1 \sqsubseteq F_2 \Rightarrow \mathcal{G}(F_1) \geq \mathcal{G}(F_2). \quad (1)$$

We define $GEN(F, e)$ as:

$$GEN(F^P, e) \triangleq \{F' | F' \sqsupseteq F_e \wedge F' \sqsupseteq F \wedge \mathcal{G}(F') > \theta\},$$

that is, the set of all features more general than both F and F_e, for which the value of $\mathcal{G}$ is greater than θ. Because by generalizing some information is lost, a pair of objects should be generalized by the most specific feature that subsumes both:

$$F' = \operatorname*{argmax}_{F'' \in GEN(F^P, e)} \mathcal{G}(F''). \quad (2)$$

The feature F' generalizes F and F_e by losing as little information as possible.

The meaning of θ is to keep F above a minimum of specificity: each generalization F' of F^P will be such that:

$$\theta_F < \mathcal{G}(F') \leq \mathcal{G}(F).$$

There is therefore an invariant during the whole training process:

$$\forall F^P = \langle F, \theta \rangle \in P_C.\mathcal{G}(F) > \theta. \tag{3}$$

3.7.2 The specialization of features

The more a feature is specific, the less likely its chance of being ambiguous is. This is because, since concepts are ordered like the respective features, the more a concept is specific, the narrower its meaning is, and more certain its classification.

If we consider two categories labelled as "Soccer" and "Tennis", we might be uncertain on where to place the concept "game". On the other side, the more specific concept of "soccer game" belongs definitely to the first category.

An ambiguous feature F should be substituted in the profile of the category by a set of more specific features. If F is ambiguous, then we remove it from the profile and add a set of features $\{F_1, \dots, F_h\}$ such that $\forall i \in \{1, \dots, h\}.F_i \sqsubset F$ (every F_i is *strictly* more specific than F).

Iterating the specification procedure for each ambiguous feature can be misleading. Steels, in [28; 29], observed that by iterating the process one can run the risk to obtain a meaningless feature that is not ambiguous just because it is too specific. Whether or not to specialize an ambiguous feature (instead of simply removing it) is therefore an issue.

We take the *necessity* of a feature into account by defining a predicate $NEC(F^P, C)$, which we will use to decide whether to specify the ambiguous feature or to delete it. Let $F^P = \langle F, \theta \rangle \in P_C$; we define:

$$\begin{aligned} NEC_e(F^P, C) \triangleq \{e \in E_C | e = \langle F_e, L_C \rangle \wedge F \sqsupseteq F_e \wedge \\ \nexists \overline{F}^P = \langle F', \theta' \rangle \in P_C \setminus \{F^P\}.F' \sqsupseteq F_e\}. \end{aligned}$$

In words, this is the set of examples examined so far for the category C for which F is the only feature in the profile that subsumes them. The definition of $NEC(F^P, C)$ will then be:

$$NEC(F^P, C) \triangleq NEC_e(F^P, C) \neq \emptyset,$$

that is, a feature is necessary when at least one of the examples for the category are subsumed only by that feature in the profile. If we deleted F^P, we would lose all traces of such an example in the profile of the category, which would be like "forgetting" it.

Different definitions of NEC are possible. Alternative definitions could require the presence of at least k examples in NEC_e, or the frequency of the feature in the collection could be considered in order to get rid of the most infrequent ones.

The elimination of unnecessary features is a way to eliminate those features that, although generated from valid examples, marked out irrelevant parts of them instead of meaningful ones. As an example, we can consider the following case: we examine two documents for the category "Music": "Van Morrison home page" and "Jimmy Page unofficial site". A generalization would result in the feature "page", which is undesirable. Hopefully, the term "page" will turn out to be ambiguous.

The choice of the F_i features that must substitute the ambiguous feature F in the profile of the category is arbitrary, as long as:

1. All F_i are more specific than F;

2. if $F^P = \langle F, \theta \rangle \in P_C$ has to have F specialized by $\{F_1, \ldots, F_h\}$, it must hold that:

$$\forall e = \langle F_e, L_C \rangle . (e \in NEC_e(F, C) \Rightarrow \exists F' \in \{F_1, \ldots, F_h\} . F' \sqsupseteq F_e).$$

The second condition ensures that the specialization is based upon the examples that originally generated F, and that no example for which F was necessary is lost.

To make sure none of the F_i is generalized down to F again when other examples are examined, as $F^P = \langle F, \theta \rangle$ is removed from the profile, the profile features $\langle F_i, \mathcal{G}(F) \rangle$ are inserted. Because of the invariant (3), none of the newly inserted features will be generalized to F again.

To give an example of generalization, we assume we are using features of depth 1. We will also assume that the subsumption relationship is equal to the "is sublist of" relationship. Specialization could simply be done by adding terms to the list of terms representing the feature. Adding terms is equivalent to add context in which to interpret the terms, and therefore narrows the meaning, making it more specific, as e. g. discussed in [27].

For features of depth k, specialization can be done by adding to the list a substructure of depth $k-1$ or by making one of the substructures more specific.

When an ambiguous feature is no longer specializable, we can decide whether or not to remove it, be it necessary or not. If its ambiguity cannot be resolved, it probably means it was generated from a bad example anyway.

3.7.3 Description of the algorithm

Here we define a parametric family of algorithms. We first make an explicit list of the parameters, of which an actual definition must be given in order to instantiate the algorithm:

- **the subsumption relationship** $\sqsupseteq$;
- **the definition of the ambiguity of a feature**;
- **an initial threshold** Θ to assign to the features when a new profile-feature is created from an example;
- **the generalization procedure**, which must be procedurally defined;
- **the specialization procedure**, same as above;
- **a predicate** $Captures(P_C, e)$, whose meaning will be obvious in the following.

These are the algorithm steps:

1. $P_C \leftarrow \emptyset$, $E_C \leftarrow \emptyset$ for each category C;

2. let $e = \langle F_e, L_C \rangle$ be a new example. For each category C in L_C: $E_C \leftarrow E_C \cup \{e\}$; generalize the features in P_C as in 2 so that they subsume F_e;

3. if for some profile P_C $(C \in L_C)$ $Captures(P_C, e)$ is false, then $P_C \leftarrow P_C \cup \langle F_e, \Theta \rangle$ and all examples in E_C are compared to F_e which is generalized if possible;

4. all ambiguous features are specialized; go back to step 2 if more examples are available.

The definition of $Captures(P_C, e)$ is not provided here, being a parameter of the algorithm. Its meaning is to decide whether the profile of a category "captures" the new example or not. If not, then the profile is extended by adding a new feature subsuming the example. A simple definition of $Captures$ would be: $\exists F^P = \langle F, \theta \rangle \in P_C.F \sqsupseteq F_e$, where F_e is the feature of the example e. With this definition, if at least one of the features in the profile can be generalized to subsume the new example, then no feature is added to the profile. A different and more strict definition could ask for the example to be subsumed by more than just one feature, or for a significant part of the example to be represented in the profile.

If every new example were "captured" by the profile, no new concept would be learned anymore: the training would have reached a fixpoint. Instead, when an example is considered "new enough", a new profile feature is added to the profile, the feature of which being equal to the example, and with an initial threshold value: Θ. The new feature will be generalized with the past examples examined for the category. In this phase, the *order* in which the old examples are matched against the new feature plays a role: sorting algorithms to maximize performance will be discussed later.

The value of Θ represents the minimum specificity allowed for a feature as measured by $\mathcal{G}$. Because every feature must have a value of $\mathcal{G}$ greater than its threshold, the higher Θ will be, the more specific features will be extracted. On the other side, when Θ has low values, very general features are allowed to form.

Step 4 ensures that every feature is "distinctive". Ambiguous features are not allowed in the category descriptions, and are made more specific in order to resolve the ambiguity.

After the examination of an example the system reaches a quiescent state with respect to feature ambiguity. The evolution of the state of the system can be seen as the evolution of a system whose balance is affected by some external event (the examination of a new example). From this unstable situation, the system takes itself into a stable state which will be different from the previous one if the perturbation was strong enough. The degree of perturbation for each new example can be interpreted as a *learning rate*, and it is controlled by the definition of the predicate $Captures$. The hardest it will be made for a profile to "capture" a new example, the more perturbations it will have to undergo.

Steps 2 and 3 do a form of unsupervised learning inside the profile of a category. The system searches for common features amongst the example of a category with no knowledge of what is relevant and what is not. Step 4 checks that the two steps above do not go too far with ambiguous features.

The process is not strictly incremental. The necessity to retain past examples is imposed by the specialization procedure, and can be desirable to better generalize a new feature. Still, the examination of a new example does not cause a complete recomputation over all examples, as it would in a pure batch system, but only a "local" adjustment. For this reason, the model could be thought of as a *life-long learning*, receiving examples e. g. in chronological order, through a time aware stream.

4 Model instantiation

Having defined a general learning scheme, we will proceed defining and testing some of its possible instantiations, showing also some techniques that can be applied to boost the quality of the results. Most of the choices made at this point were made without classification in mind, trying, instead, to explore new and different techniques that we thought would have suited our model. The reason of such a choice was also meant to show the flexibility of the proposed model.

The most significant step to proceed towards an instantiation is the choice of the representation for documents. We decided to exploit some of the more recent and cutting-edge experiments, and adopt some newly introduced document representation.

4.1 Text commentaries

We considered two main alternatives, when we had to choose the kind of documents we were going to test the algorithm on. The first choice was to represent the documents by their content, as it is usually done in most classification algorithms. Because most of them come directly from information retrieval experiences, they index documents over their whole content, upon which information retrieval techniques used to work in order to give good representation of all informations in a document. This kind of representation is called representation *by content*.

For as obvious the representation by content might seem, it is not the only possible one. A shorter, and more compact description of the document, not unlike the abstracts typically found at the beginning of scientific articles, would be as effective, when it comes to classify a document by the topics it treats. If not all information given by the actual document is present, the main topic is for sure. Similar representations are used by Attardi et al. in the Theseus system [3], in which categorization is done by considering the *context* a link to a document appears in, rather than the document itself. The assumption made by Attardi et al. is that the information around the link is sufficient for the user to determine the relevance of the linked document, and can then be used for classification/retrieval tasks. In many cases, the context around the link can give an explicit description of the pointed document, not unlike the kind of description an abstract gives about the content of an article. In this case, the description is less likely to contain metaphors, digressions, non relevant parts to it, and can be very effective for classification purposes.

Other than Theseus, this kind of descriptions have also been used to evaluate the performance of keyword or feature extractors in [22; 6].

The representation by content has without doubts several "pros". Many collections, such as the Reuters, OHSUMED or MEDLINE collections are available as standard tests for classifiers, and can therefore be used to compare the performance of different systems, as in [34]. These collections are formed of textual documents pre-categorized by experts of the domain (for instance, the Reuters collection contains 21578 press agencies, while OHSUMED and MEDLINE contain more than 300.000 medical articles). Because these collections are fairly large and are based on real documents, they show the performance of the system in a real domain. One of the disadvantages is that the specificity of the domain itself can give misleading indications about the behavior of the system in a different environment. For our special goal, also, it would be difficult to judge the quality of the description generated for each category without the help of an expert.

With the use of shorter descriptions, or representation by context, texts are shorter, thus making the learning process ask for fewer resources and making prototyping easier. While the limited amount of information provided by such descriptions limits the chance of irrelevant information, on the other side it makes frequency based considerations worthless, due to the almost unique occurrence of each term in each description. This kind of representation is very well suited for the representation of web pages, especially the main sites ones, since they often contain non textual informations such as images, that are hard or impossible to index. In the case of the catalog organized search engines (the ones built manually), we can use the "blurb" they give for each document as the document description, thus easily obtaining a large and "real" collection of documents to test our system on. This allows us to classify documents in languages other than English, for which languages standard collections are harder to find, and evaluate

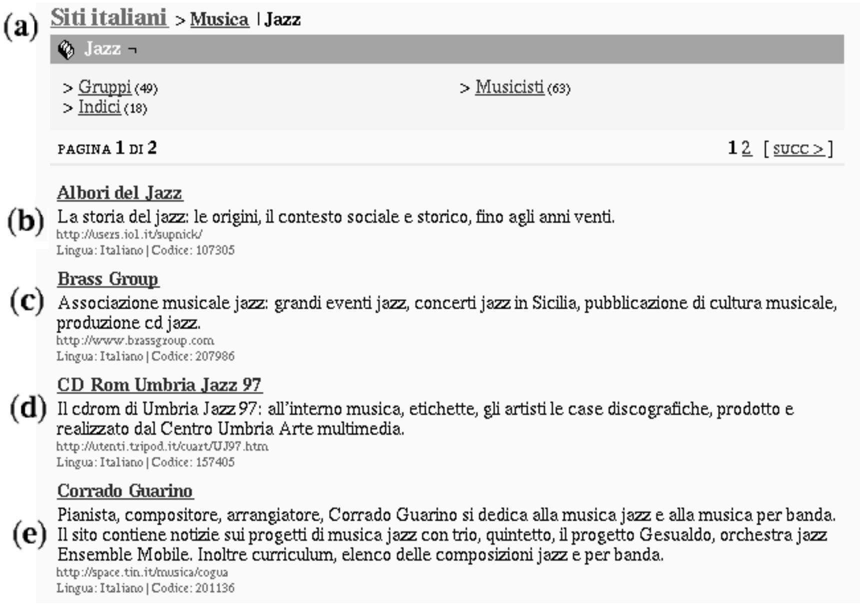

Figure 4: **excerpt from a Virgilio page: from this page we obtain the category name (a) and four commentaries (b), (c), (d) and (e) as examples of the category.**

the quality of the descriptions obtained easily, being the catalogs organized in taxonomies with common sense meaning.

Many search engines give a short description of some kind of the retrieved document (e. g. [1; 5; 2]), to help the user visiting only relevant pages. From some of the engines organized in catalogs, such as Yahoo.it and Virgilio [32; 31], it is possible to automatically obtain a set of descriptions called *commentaries* with the relative classification.

The main disadvantage is that there is no standard collection and confrontations with other classifiers are then difficult.

We chose to use commentaries, since classification was just a secondary goal, and since commentaries allow us to experiment more about the feature structure relatively to the document structure: feature structure can be made equal to the document representation. Also, the techniques described here could be in the future integrated in Theseus, to explore new means of classification and learning.

To extract the commentaries we used the *collector* of Theseus: its "spider", which crawls inside a site (such as Yahoo), and extracts by a filter the commentaries we need. This way we obtained approximately 22.000 commentaries from Virgilio's catalog, which were used in the training and test processes.

4.2 Noun phrases

We identify noun phrases inside the text of the commentary, using basic linguistic analysis. Formally, a noun phrase in Italian is made of nouns and modifiers such as adjectives and adverbs. Noun phrases contribute to the meaning of a text with the verb phrases [23].

Noun phrases can have several roles in the text, but they mainly identify the subject and object of the sentence. The use of noun phrases has been studied as a mean to achieve better results than by using simple words [13; 17]. Noun phrases are seldom ambiguous, since the intrinsic ambiguity of terms is reduced by the context of the other terms of the phrase, which help specify each other's meaning. Noun phrases express therefore more detailed and narrow concepts than single terms.

In information retrieval, many works have exploited noun phrases as an alternative to terms for document indexing. One of the more recent is the *SIMPR* project (Structured Information Management: Processing and Retrieval), part of which tries to evaluate the opportunity to represent text using natural language processing. The project approaches the problem observing that some concepts are only expressible by the use of noun phrases, and that the use of terms can therefore be inappropriate in some case.

4.3 Document structure

Commentaries representing the documents have then be reduced to the list of noun phrases they contain, discarding verbs, conjunctions, prepositions and keeping nouns, proper nouns, adjectives, adverbs and past participles (considered as adjectives). The document representation is therefore a list of noun phrases, whose depth is 2.

4.4 POS tagging: TreeTagger

In order to extract noun phrases from a text, we must first identify the grammatical class of each term.

This task is called *Part of Speech Tagging*, since the grammatical category is identified by attaching a tag to each term. Tagging is not only useful to identify noun phrases: sometimes it can help disambiguate a term, and it represents additional information, which is stored outside the documents.

A tag can belong to different syntactical categories depending on the context it is in. The tagging algorithm, if sophisticated, will also proceed to identify which tag is correct for the term. Many times, for example, an adjective can be used as a noun. To assign tags, it is therefore a good approach to examine the whole period.

To extract the noun phrases from documents we used *TreeTagger*, a statistic linguistic tool based on the theory proposed by Schmid in [26], developed at the Institute for Computational Linguistic of the University of Stuttgart. The italian lexicon and training examples were made available by Achim Stein. The tagger has been improved and made available as server by Giuseppe Attardi; it is freely available and queries can be easily sent by a Java client. Another class has the task to form noun phrases relying on the tags returned by the tagger.

The technique used in the TreeTagger is similar to the one used in other analogue tools based on n-grams. In both cases, the probability of a sequence of terms/tags is estimated, by the following recursive formula:

$$p(w_1 \dots w_n, t_1 \dots t_n) = p(t_n |\ t_{n-2}t_{n-1}) p(w_n |\ t_n) p(w_1 \dots w_{n-1}, t_1 \dots t_{n-1}),$$

where $w_1, \dots, w_n$ is a sequence of words, and $t_1, \dots, t_n$ are the relative tags. $p(w|t)$ is the probability that the word w has tag t.

Schmid uses a particular technique to estimate the probability of a tag given the two tags before it: $p(t_n |\ t_{n-2}t_{n-1})$. The most used technique is called "maximum likelihood estimation" and is expressed by the formula:

$$p(t_n |\ t_{n-2}t_{n-1}) = \frac{\mathcal{F}(t_{n-2}t_{n-1}t_n)}{\mathcal{F}(t_{n-2}t_{n-1})},$$

where $\mathcal{F}(t_{n-2}t_{n-1}t_n)$ is the number of occurrences of the tri-gram $t_{n-2}t_{n-1}t_n$ in the collection, and $\mathcal{F}(t_{n-2}t_{n-1})$ is the number of occurrences of the bi-gram $t_{n-2}t_{n-1}$.

Schmid points out that this estimation is affected by the low frequency of some of the data it relies on, and whose estimation could therefore be inaccurate.

To avoid this problem, TreeTagger estimates the aforementioned probabilities by constructing a binary decision tree. The probability of a tri-gram is obtained by following the relative path down the tree, reaching a leaf.

The construction of the tree is recursive, starting from a set of known tri-grams as training set. At every step, a test to split the original set so that the probability distribution that the third tag (t_n) is well split is constructed. Tests are in the form:

$$tag_{-i} = t; \quad i \in 1, 2; \quad t \in T$$

where T is the set of tags. All tests are evaluated, by evaluating the amount of information acquired by the third tag. A threshold strategy is defined to stop the construction of the tree.

A pruning phase is implemented, to prune out branches for which the information is under a threshold.

The lexicon used by Schmid contains the probability of each tag for every word. The lexicon has been divided in two subsets, the first one containing data about actual terms, while the second containing information about the suffixes, in order to make the system able to tag terms it has never met before. The longest suffix is five letters long.

The accuracy of Schmid's method is reported to be higher than the one achieved by standard techniques. The system has also proven to be more robust to variations in the dimension of the tri-gram set used for the training process.

In order to extract noun phrases we analyze the sequence of tags associated to each term in the period. This is done by a finite state automaton which recognizes the language generated by the following grammar (N_P is the start symbol):

$$\begin{aligned}
&N_P ::= Q\mathtt{N}R \\
&Q ::= \mathtt{Adj}LQ|\mathtt{Adv}LQ|\mathtt{Pp}LQ|\epsilon \\
&L ::= \mathtt{ConAdj}L|\mathtt{ConPp}L|\epsilon \\
&R ::= \mathtt{Prep}N_P|N_P|\epsilon,
\end{aligned}$$

where `Adj`, `Adv`, `N`, `Pp`, `Prep` and `Con` are the tags for adjectives, adverbs, nouns, past participle, prepositions and conjunctions respectively. When another tag is found, or the sequence is no longer parsable through this grammar, then the noun phrase ends.

This is not the only way to define or identify noun phrases. Different or more accurate definitions are possible, for example treating conjunctions differently, or by a more accurate linguistic analysis.

Shown here is a commentary (in Italian) of a document belonging to the category labeled as "Sport":

Federazione Italiana Pallavolo (FIPAV):

Le novità, i comunicati stampa, l'attività agonistica delle nazionali e dei club. Una rubrica dedicata al beach volley.

The tagger extracts the following noun phrases:

```
((FEDERAZIONE__NOM ITALIANO__ADJ PALLAVOLO__NOM)
 (FIPAV__NOM)
 (NOVITA__NOM)
 (COMUNICATO__NOM STAMPA__NOM)
 (ATTIVITA__NOM AGONISTICO__ADJ NAZIONALE__NOM)
 (CLUB__NOM)
 (RUBRICA__NOM DEDICARE__VER-PPER BEACH__NOM VOLLEY__NOM))
```

Noun phrases are represented each one on a different line, with their tag concatenated to the term after "__". The meaning of each tag is reported in figure 5.

tag	*grammatical category*
`NOM`	noun
`NPR`	proper noun
`ADJ`	adjective
`ADV`	adverb
`VER-PPER`	past participle

Figure 5: **tags and relative grammatical categories.**

The disambiguation operated by TreeTagger is noticeable for the term `NAZIONALE`, which is an adjective in Italian, but which is used as a noun in the sentence.

4.5 Feature structure

Given the representation chosen for the documents, different structures can be used for the features.

Features of depth 2 allow the expression of concepts by using multiple noun phrases. We represented noun phrases as *unordered* lists of terms, assuming that the order does not play a role. In many practical cases, expressions in which the order of words is changed have the same meaning, like for the two noun phrases "information retrieval" and "retrieval of information". In general, though, order does play a role, and our assumption is therefore an approximation.

To use the noun phrases as features, the structure must be of depth 1. Concepts will be "approximated" by noun phrases, and the specialization of a concept will be done by adding terms to the list that represents it. The added terms will be considered as specifiers, such as adjectives, extra complements and the like. Each noun phrase in the training corpus of documents will be used as a training example for the category of the document it appears in. As we have discussed above, since not all noun phrases are good examples for the category of the document that contains them (actually, only a small number will be), we are making a strong approximation of the truth.

No information about the co-occurrence of two noun phrases will be kept, discarding a significant amount of information. With respect to the case of lists of noun phrases as features, we must note that the examples will be more, since generally a document will contain more than one noun phrase. The quantity of examples will become an issue, and it will therefore be necessary, as we "drop" information on the analytical side, to take it back on the statistical side, weighting the features appropriately so as to discard the irrelevant ones.

The use of simple terms as features can still be experimented with the representation we gave of documents. The structure of a document will mostly be unused, as only the terms inside of it will be of interest in this case. The features will have depth 0.

4.6 Examples of documents and features

We will show with a few examples the various kind of features that a document can be considered with.

We will use as an example the following commentary (in Italian):

FESTIVAL DI SANREMO

Sito dedicato al Festival di Sanremo, la più importante rassegna mondiale della canzone italiana. Il sito contiene anche un interessante database con il quale è possibile ottenere tutte le informazioni storiche su canzoni e cantanti. È inoltre possibile ascoltare i successi più gettonati, in real audio.

The commentary is relative to a document classified under the category `MUSIC` in Virgilio's catalog.

From the commentary, the system will extract the following noun phrases:

```
(FESTIVAL__NOM SANREMO__NOM SITO__ADJ DEDICARE__VER-PPER)
(PIU__ADV IMPORTANTE__ADJ RASSEGNA__NOM MONDIALE__ADJ
 CANZONE__NOM ITALIANO__ADJ)
(SITO__NOM)
(INTERESSANTE__ADJ DATABASE__NOM)
(INFORMAZIONE__NOM STORICO__ADJ CANZONE__NOM)
(CANTANTE__NOM)
(SUCCESSO__NOM PIU__ADV GETTONARE__VER-PPER)
(REAL__NOM AUDIO__NOM).
```

If we choose lists of noun phrases as features, with depth 2, this list is the only feature that will be considered for the document. There will be a one to one correspondence between documents and training examples.

If we use noun phrases as features, whose depth is 1, then each noun phrase will be a feature, and hence a training example: eight examples would be obtained from the document above. It is evident that some of the features are actually not connected to the category meaning at all, such as `(INTERESSANTE__ADJ DATABASE__NOM)`.

If we decide to use simple terms, features with depth 0, each term will be a feature and a training example.

4.7 The ordering over features

The definition of the order relationship over features represents the second main step towards an instantiation. The definition of the relationship will be given for the three different cases we discussed: lists of noun phrases, noun phrases and terms. As a result, three different systems will be obtained, each one with a different behavior.

4.8 Ordering over terms

We do not define any relationship over simple terms. A definition would be possible by using some sophisticated linguistic tool with a large database of terms, but since we want to stick to the syntax, and terms are for us atomic, we can just assume that every different term is unrelated. If terms were not atomic, if they were for instance of depth 1, some form of relationship could be defined. Although this has not been done in the complete system, it is worth, as a curiosity and as a digression that will show the flexibility of the model, to summarize what the behavior of the system would be in that case.

We could represent terms as depth 1 features, using letters as atomic features (depth 0). Because we can compare the elements of the sequence that makes terms, we could then define a subsumption relationship as follows (t_i being terms): $t_1 \sqsupseteq t_2$ iff t_1 is a prefix of t_2. *Word stemming* is the technique to cut out the suffixes of terms in order to put together all terms that share the same linguistic root, in the assumption that their meaning will be very close. Words are, in most languages, composed of roots and suffixes: the root gives the word meaning, while the suffix gives it its role in the phrase. For example, the words "stock", "stocking", "stocks", "stocked" share the same root "stock-", and refer mostly to the same concept of "stocking".

While the root of words has etymological reasons, if we simply interpret it as a prefix of the word, we could say that the root "st-" is more general than "stock-", since it is a prefix of it. The excessive reduction of roots is known before systems that use word stemming as *over-stemming*, and it is cause of ambiguity. By considering the prefix "st-", we would have all of the above terms put together with "stuck", "stick" and "stack" and variations, which have very different meanings. On the other hand, an overly long root would split the concept too much: with any root longer than "stock-", all of the examples above would fall in a different category. This phenomenon is called *under-stemming*.

The over-stemming problem can be seen as an excessive generality (ambiguity of roots), while the under-stemming is an excessive specificity.

Suppose now that we instantiate our model with the aforementioned scheme: we would obtain a system that, given some example of unstemmed terms grouped under the category of the same meaning, would build for each meaning a set of roots that are not as general as being ambiguous (in the domain the training is brought on), but that are as general as possible. The roots the system would output would not necessary be the linguistically correct ones, but would nevertheless have the required characteristics.

4.8.1 Ordering over noun phrases

Noun phrases are represented by a structure of depth 1. Because a noun phrase is a list of terms, it is natural to compare the terms contained in the lists representing the two features. By assumption, we observe that the more terms appear in the phrase, the more the meaning of each term will be clear, since more context will be available to disambiguate it. By the way we formed noun phrases, for instance, we notice that a specification complement would be part of the same noun phrase as the complement it refers to, thus suggesting that the noun phrase containing more terms is more specific.

Still, in some cases, depending on the terms, many terms can be vaguer than a shorter sentence ("the fact that this difficulty has arised..." is vaguer than "this problem..."). Still, when the terms are the same (or almost), then it makes sense to say that the longer sentence is more specific. In a nutshell, indicating by $\sqsupseteq_1$ the subsumption relationship between features of this kind, we define:

$$F_1 \sqsupseteq_1 F_2 \triangleq F_1 \subseteq F_2.$$

When two features contain a term each which does not occur in the other, they will be considered unrelated.

By representing features of depth 1 in the usual way, we have that the feature:

```
(FESTIVAL_NOM CANZONE_NOM)
```

is more general than:

(FESTIVAL_NOM CANZONE_NOM ITALIANA_ADJ)

but more specific than:

(CANZONE_NOM).

4.8.2 Ordering over lists of noun phrases

Lists of noun phrases have depth 2. The definition of the subsumption relationship can generalize the definition we just introduced for features of depth 1.

Consider 2 features F_1 and F_2 of depth 2 containing k and h noun phrases respectively:

$$F_1 = \{np_1^1, np_2^1, \ldots, np_k^1\}$$
$$F_2 = \{np_1^2, np_2^2, \ldots, np_h^2\},$$

where np_j^i is a noun phrase.

We will say that:

$$F_1 \sqsupseteq_2 F_2 \triangleq \exists M.\ (\forall (i,j) \in M.\ np_i^1 \sqsupseteq_1 np_j^2) \wedge (\forall i \leq k.\ \exists! j.\ (i,j) \in M) \wedge$$
$$((i_1, j) \in M \wedge (i_2, j) \in M \Rightarrow i_1 = i_2).$$

In words, we impose that each noun phrase of F_1 be more general (with the definition we used for feature of depth 1) of a distinct noun phrase of F_2. F_2 must therefore contain more noun phrases than F_1 ($h \geq k$). In the formal definition, M is a set of pairs of indexes that associates each noun phrase of F_1 to a distinct and more specific one in F_2.

Features are represented by trees of depth 2. The feature F_1 will then be more general than F_2 if the tree that represents it can be obtained from the one representing F_2 by pruning some branch. This situation is better shown and exemplified on figure 6, in which the features ((A B C D) (E F) (G H I)) and ((A C D) (G I)) are compared.

Because our documents are also represented as structures of depth 2, we can interpret (a), in the figure, as a document, and (b) as a feature. (c) shows us how (a) pertains to (b).

4.8.3 The function $\mathcal{G}$

The function $\mathcal{G}$, first introduced in 3.7.1, is required to make sure features do not get too general during the process. In the case of terms, depth 0 features, because no generalization will ever take place, $\mathcal{G}$ will not play any special role. Still, because it is procedurally needed that the value of $\mathcal{G}$ for any feature be higher than its threshold, we will have to define $\mathcal{G}$ as a constant higher than the initial threshold Θ, for example $\Theta + 1$.

When features are more structured, and have hence depth 1 or 2, the generalization/specialization processes require a non trivial definition of $\mathcal{G}$. The only requirement for $\mathcal{G}$, however, is the one described by 1.

Consider the number of terms contained in the description of a feature as our $\mathcal{G}$ function: we will verify that this definition suits our purposes well.

Suppose that condition 1 is not valid by absurd, that is, let us assume $\mathcal{G}(F_1) < \mathcal{G}(F_2)$. This means that the feature F_1 contains fewer terms than F_2. If F_1 and F_2 have depth 1, this violates, by definition of $\sqsubseteq$, the hypothesis that $F_1 \sqsubseteq F_2$. If features have depth 2, because every noun phrase of F_2 should be contained in a different noun phrase of F_1, the latter could not contain fewer terms, and then the hypothesis is violated again.

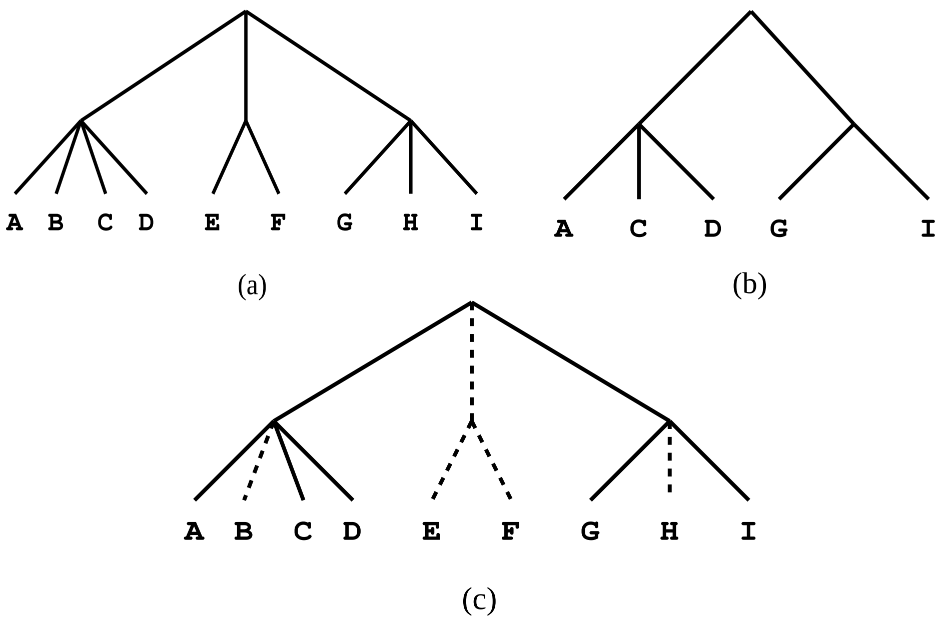

Figure 6: **an example of subsumption between features of depth 2. The feature (a) is more specific than feature (b). In (c), (a)'s extra branches are marked by dashed lines.**

As a conclusion, the number of leaves in the tree representing a feature is a good definition of $\mathcal{G}$.

More sophisticated $\mathcal{G}$'s could be defined, for example considering the tag of the terms contained in the feature description. In particular, because adjectives and adverbs are subject, inside the noun phrase, to the presence of nouns, one could give nouns a higher value than to adjectives or adverbs. As a result, one would rather obtain a list of two nouns instead of a list of a noun and many adjectives.

To obtain this result, we have defined $\mathcal{G}$ as the number of nouns plus the number of adjectives and adverbs times 0.1.

With the definition we just gave, the requirement imposed by 1 is still valid. A feature F_1, to be more specific than F_2, still needs to have a superset of F_2's terms, and its $\mathcal{G}$ will therefore be greater.

4.9 The generalization procedure

For the features of depth 1 the generalization is obtained in a trivial way. In ART, input vectors and prototypes represent lists of features (1 if present, 0 if not); similarly, in our case, they are made of lists of terms. In ART, generalization was done intersecting the prototype with the incoming vector, so that only the features common to the whole cluster would appear in the prototype.

In our case, is is sufficient to intersect the lists of terms the features are made of in order to generalize, since the sublist that will be obtained will subsume both the lists that generated it. The intersection is by definition the generalization that maximizes $\mathcal{G}$, and will therefore substitute the feature in the profile, provided its $\mathcal{G}$ is higher than θ of the profile feature.

For the features of depth 2, the extra level in the structure makes things more complicated. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ the functions $\mathcal{G}$ for the depth 1 and 2 respectively, as defined above.

From the definition of $\mathcal{G}_2$, we have:

$$\mathcal{G}_2(\{np_1, \ldots, np_h\}) = \sum_{i=1}^{h} \mathcal{G}_1(np_i),$$

where np_i are noun phrases.

To find a feature F' that subsumes $F = \{np_1^F, \ldots, np_h^F\}$ and $F_e = \{np_1^e, \ldots, np_k^e\}$ we look for the set M of pairs of indices $\langle i, j\rangle$, which connects noun phrases of F to the ones of F_e so as to maximize:

$$\mathcal{G}_2(\{np_i^F \cap np_j^e\}_{(i,j)\in M}) = \sum_{(i,j)\in M} \mathcal{G}_1(np_i^F \cap np_j^e).$$

It is a maximum cost matching problem, which can be approached as follows: we search for the matching M amongst the noun phrases np_i^F and np_j^e. The cost of every pair is $\mathcal{G}_1(np_i^F \cap np_j^e)$.

The matching obtained allows us to determine a feature subsuming both F and F_e. If $\mathcal{G}_2$ is greater then the threshold of the profile feature of feature F, it will be generalized. To solve the matching problem we used an algorithm based on the shortest path tree algorithm.

4.10 Ambiguity of features

The control for ambiguous feature allows the system to do a sort of "backtracking", once some generalization has proven to be inadequate or excessive. For this reason, the ambiguity of features will determine its discrimination quality.

4.10.1 Ambiguity as excessive generality

We want the concepts extracted for the documents in order to describe a category to be "distinctive". We expect them to identify the category they refer to without any doubt or uncertainty: formally, we want the concept in a profile not to subsume any concept in a different profile.

For as strong this assumption may sound, it makes the system identify the most peculiar characteristics of any category; still, a document, in principle, can belong to any number of categories according to the descriptions given, since nothing keeps it from pertaining features in different profiles.

We will say that a feature F is ambiguous when:

$$Ambiguous(F) \triangleq F \in P_C \wedge \exists F' \in P_{C'}.C \neq C' \wedge F \sqsupseteq F'.$$

If F is ambiguous, then it expresses a concept that can be specified so as to determine its belonging to different categories, and it is therefore not adequate to be in a category profile.

We can take Virgilio's catalog structure as an example. Consider the categories "Trips and Tourism" and "Environment and Nature". Suppose that the feature of depth 1 "city plant" belongs to the former category. If a feature such as "plant" is in the description of the latter category, then it is ambiguous, because one of its specification belongs to "Trips and Tourism".

4.10.2 Non specializable features

There are modifications, special cases or improvements, that can be made to the schema we just described.

Consider the case of depth 0 features, that is, simple terms. Because depth 0 feature cannot be specified, its ambiguity would determine its elimination. Because it is unlikely to find unambiguous depth 0 features, we would rather, in this case, allow features to be ambiguous, and blindly trust the weighting measure to mark out the most important features. This way, no specialization will be performed: we will therefore say, in this case, that:

$$Ambiguous(F) \triangleq \perp.$$

4.10.3 Non disjointed classes

For more structured features, like depth 1 or 2 features as we considered, the characteristics of the domain should be taken into account when defining the ambiguity predicate. It is especially important to take care of the partial overlap that categories might have amongst them.

The description of a category that is built by the system depends on the context the training process is done, in the sense of the other classes contemporaneously learned. The ambiguity of concepts will also depend on that factor, so that a feature can be ambiguous or not in the same profile depending on which other profiles are there. Suppose we are learning the descriptions of the categories labeled as “Sport”, “Music” and “Politics”. The concept overlap amongst these categories is discardable, and it is therefore reasonable to look for a distinguishing set of concepts to describe each category. The ambiguity of a concept would be due to its irrelevance to the category (as it would happen for a concept such as “home page”) or to its extreme generality, as it could be for a concept such as “event”, which could belong to either one of the three categories.

Suppose now we are learning the descriptions of categories labeled as “Music”, “Jazz”, “Blues”. The overlap amongst these categories is not to be discarded, as it is significant for “Jazz” and “Blues”, and total for the same two relatively to “Music”. Semantically, every concept good enough to describe “Music” would make it, if specified, to the category “Jazz”. Even assuming that we always classify documents in the more specific class, and that the categories are a partition of the examples, we still would not obtain very good descriptions.

What we do in this case, is to say that a feature is *not* ambiguous when the feature and a more specific one, in the other profile, share some of the documents. If we train the system telling it that every “Jazz” document is also a “Music” one, we would as a result obtain a description of “Jazz” which is exactly a specification of the one of “Music”.

When two categories share some of the documents, this can be because of concepts that semantically belong to *both* categories. As an example, consider the categories labeled as “Jazz” and “Blues”: these two could share a document talking about a concert of an artist that used to play both Jazz and Blues music. The two categories do not share the document because it talks about distinct concepts relative to Jazz or Blues, but because it talks about concepts that are actually *shared* and belonging to both. When it comes, for example, to the name of an artist, no further specification (not an easily expressible one) would help discriminating the two categories. We will then assume that when two categories share a document for the same reason (for the same, or a couple of subsuming features), then none of the features involved is ambiguous for that reason.

This “corrective” can introduce some noise due to random factors; in particular, there’s a risk that with this definition it is no longer possible to apply specialization as a mean to eliminate inadequate and unnecessary features. Because unnecessary or meaningless features can be shared amongst categories, the risk is real. It is a consequence of the fact that the task of learning descriptions for non disjointed classes is harder than for disjointed classes.

4.11 The specialization process

The specialization algorithm will be composed of the following steps, where $F^P = \langle F, \theta \rangle$ is the profile feature whose feature is to be specified, C is the category to which it belongs:

1. $E \leftarrow NEC_e(F^P, C)$; $\mathcal{F} \leftarrow \emptyset$;
2. Extract $e = \langle F_e, L_C \rangle$ from E. Generalize the features in $\mathcal{F}$, if possible, so as to subsume F_e;
3. If $\nexists \overline{F}^P = \langle F', \theta \rangle \in \mathcal{F}.F' \sqsupseteq F_e$, then $\mathcal{F} \leftarrow \mathcal{F} \cup \{\langle F_e, \mathcal{G}(F) \rangle\}$; the new feature is generalized against the examples in $NEC_e(F, C)$.

   If $E \neq \emptyset$ go to step 2;
4. $P_C \leftarrow P_C \setminus \{F^P\} \cup \mathcal{F}$.

In words, we consider the set of examples for which F is necessary. On these examples, we do a "mini-training", imposing the formation of feature more specific than F; for this purpose, their specificity threshold is put to the value of $\mathcal{G}(F)$. Since all examples in the training set share the feature F, this will be in all extracted feature, which will therefore be more specific than F.

The steps 2 and 3 of these algorithms are similar to the steps 2 and 3 of the main training algorithm described in 3.7.3.

4.12 The predicate "*Captures*"

To proceed with the instantiation of the algorithm, it is necessary to define the condition under which a new example causes the creation of a new feature: the predicate "*Captures*".

If we think of classification, this predicate could be seen as a classifier. Taking the profile of the category and the new example, it would determine whether the example was correctly classified by the profile, and leave the state unchanged in affirmative case. If the example is instead misclassified (it is said not to belong to the category), then a modification to the profile is made, in order to make it "recognize" the new example.

For many reasons, however, this is either impractical or incorrect. From a classification point of view, this situation would be desirable, and it could be seen as an internal evaluation scheme not unlike the one used by "wrapper" learning algorithms. These methods reserve part of the training set to test their improvement right after a change is made to the state, in order to detect a decay in performance. To be effective, however, classification needs to have weights associated to the features, and it would be highly impractical to evaluate them all at learning time.

From our viewpoint, a solution as the one proposed would not only be impractical, but also not totally correct. Such a solution would privilege the retrieval over the quality of representation, and is it well known that, in some cases, high accuracy in classification could be done by unexpected features, usually because of idiosyncrasies of the domain.

Our goal is to identify in an example the concepts that are a category's topics. For this reason, we will prefer a method that simply identifies the features inside the profile in the document, which is known to belong to the category. If such features are not found, then the document belongs to the category for unknown reasons, and features must be added to capture the new case.

We decided that an example is to be considered "new" (thus the *Captures* predicate is false) when fewer than 10% of the terms in the text are found in features in the profile subsuming the new example. If the example contains n terms, we will say that it is a "new" one when less than $[\frac{n}{10}]$ of its terms are found in said features. This imposes that not only some feature subsumes the new example, but also some part of it is "covered" by said features. The value of 10% has been determined experimentally, and can be varied. By making it greater, more of the document will be requested to be identified in the profile, while keeping this value (which we call η) low is a more relaxed approach.

Choosing a value higher than 0% for η has pros and cons:

- The main advantage is evident when an example is identified by the profile, but for less than η. In this case, the algorithm "exploits" the example, assuming that other interesting features could be obtained from it. If said features are actually not present, some ambiguous one will be added to the profile, and will likely be eliminated later in the process.

- The main disadvantage is that it requires the system to perform an extra check. Because at generalization time some of the features in the document could have been found already, we must check that the generalization in course is not equal to one which is already member of the profile. This step is not needed when η is 0.

In our case, the value of η makes sense only when the depth of feature is 2. When the depth is 0, the profile either recognizes the term or not, while it is unlikely for a noun phrase to have more than 10 terms in it (although of course a value of eta big enough could be specified for noun phrases).

The case of features of depth 2 is different, because since a feature is the whole text, we know that it is assigned to the category for a reason, and it makes sense to try and exploit it more than if we were uncertain of whether it is actually a good example.

4.13 Initial threshold

The last thing to define in order to obtain a full instantiation is the value for the initial threshold Θ.

For simple terms, feature of depth 0, the value of Θ has no meaning, since we already decided that no proper generalization takes place. For procedural purposes, whatever Θ, $\mathcal{G}$ must be a constant higher that Θ.

For feature of higher depths, we must consider the definition of $\mathcal{G}$ to have the correct grasp on the meaning of Θ. Because $\mathcal{G}$ is defined so as to grow as the list representing the feature grows, as a consequence greater values of θ will cause the formation of longer features. If we want to limit the generality of a feature in a profile, we can then choose the appropriate value of Θ. In the instantiation we used $\Theta = 0$, so as to allow features as short as single terms. If, in general, a single term is not enough to represent a concept, in many practical cases it will be unambiguous in the context it is learned.

4.14 Typical situations during the training process

We will now show some examples of typical situations that occur during the training process.

4.14.1 Terms

The situation using simple terms is very straightforward. Every term seen in the training examples is stored in the profile of the category the example belongs to.

4.14.2 Noun phrases and lists of noun phrases

Using more structured features things get a lot more interesting and articulated. We will show these two cases together as the phenomenology of situation is very similar.

We will take generalization into account, as the first example. Suppose we have no features in the profiles, because the training process just begun. The following example, example e_1 will be examined by the system, for the category MUSICA:

```
((FESTIVAL__NOM SANREMO__NOM SITO__ADJ DEDICARE__VER-PPER)
 (PIU__ADV IMPORTANTE__ADJ RASSEGNA__NOM MONDIALE__ADJ
  CANZONE__NOM ITALIANO__ADJ)
 (SITO__NOM)
 (INTERESSANTE__ADJ DATABASE__NOM)
 (INFORMAZIONE__NOM STORICO__ADJ CANZONE__NOM)
 (CANTANTE__NOM)
 (SUCCESSO__NOM PIU__ADV GETTONARE__VER-PPER)
 (REAL__NOM AUDIO__NOM)).
```

Features have depth 2. Because profiles are still empty, e_1 will be plainly inserted into the profiles as a feature F_1 whose threshold will be set to 0.

Suppose another example (e_2) for the same category arrives:

```
((FESTIVAL__NOM SANREMO__NOM)
 (EVENTO__NOM)
 (INTERVISTA__NOM RETE__NOM)
 (FATTO__NOM)
 (FESTIVAL__NOM CANZONE__NOM SANREMO__NOM)).
```

Generalization with feature F_1 in the profile will cause F_1 to become:

```
((FESTIVAL__NOM SANREMO__NOM)
 (CANZONE__NOM)).
```

The generalization is valid because the threshold of F_1 is surpassed.

Suppose we add another example e_3 for the same category:

```
((SANREMO__NOM CLASSICO__ADJ CONCORSO__NOM
  ANNUALE__ADJ MUSICA__NOM CLASSICO__ADJ)
 (PIANOFORTE__NOM)
 (CATEGORIA__NOM)
 (SEDE__NOM SANREMO__ADJ)
 (ORGANIZZAZIONE__NOM MUSIKSTRASSE__NOM)
 (GRUPPO__NOM EDITORIALE__ADJ BIXIO__ADJ)
 (PATROCINIO__NOM COMUNE__NOM)).
```

By matching e_3 against F_1, the feature will become:

```
((SANREMO__NOM)).
```

The threshold, being 0, still allows generalization. The generality of the feature is, however, excessive, as "sanremo", in Italian, is not necessarily associated to the category MUSICA.

Consider the profile of a category labeled as SPORT, and suppose it contains the feature F_1':

```
((SANREMO__NOM))
```

which was generated by the generalization of the two examples:

```
((MILANO__NOM SANREMO__NOM)
 (GRANDE__ADJ CLASSICO__NOM)
 (CICLISMO__NOM ITALIANO__ADJ))
```

and

```
((MARIO__NPR CIPOLLINI__NOM)
 (TRAGUARDO__NOM SANREMO__NOM)).
```

In this case the ambiguity of both features is obvious, and needs specification. We will only show what happens in the category `MUSICA`, as for `SPORT` the same reasoning will apply.

The system will consider e_1, e_2 and e_3, the examples subsumed by the feature F_1. A threshold of 1 will be defined to impose the formation of more specific features. The three examples are examined again (for simplicity, in the same order): after the first two example the behavior is the same as before, but when the third example is examined, the feature `((SANREMO_NOM))` will be considered too general, and e_3 will form a new feature. Thus these two more specific features will be added to the profile instead of the ambiguous one.

5 Feature weighting

After the training phase, during which features are constructed, we apply a feature weighting phase. This phase associates a numerical weight to each feature, in an attempt to mark the more important features as opposed to the less crucial to the meaning of the category.

Although this phase is done only *after* the training algorithm has completed, it still have to be considered a learning task. The most popular learning algorithm over text, such as Rocchio's, as a matter of fact do learn only by assigning weights. Without this step, profiles would be a set of features in which it would be impossible to understand the relative importance of each feature.

Also, we must consider that some unwanted feature might have made it through the learning process, and could be identified as non relevant at this point.

Needless to say, for numerical and statistical based approaches to learning from text, this is the very main step. In these methods, a feature selection step may or may not be performed prior to feature weighting, which is the true place where the algorithm decides what belongs to the category meaning and what does not.

Some sort of feature estimation by a numerical weight is important in more symbolic based methods as well. In an algorithm such as the decision tree learning one, features have to be selected and compared when a node has to be created. The "rank" of the feature can be determined in many ways (one of which being the "wrapper" technique we discussed elsewhere) but the easiest way is to use a heuristic to evaluate the features, and select the best one.

There is nothing like the best weighting technique overall. While many reasonably tested and stable measures exist (such as TF-IDF and Rocchio), it is very possible that some exotic measure would perform considerably better than others in some specific domain. Many times a measure chosen ad hoc by study of the domain can help taking some peculiarities into account. On the other hand, many things about information retrieval and automatic categorization have already proven to be counter-intuitive (e. g. the useless noun phrase indexing), and it would not be a safe bet to go for the unexperimented.

5.1 TF-IDF weighting technique

The weighting technique called *TF-IDF* (Term Frequency/Inverse Document Frequency) is one of the most widely used techniques (e. g. from Joachims in [11]).

This technique is usually used to represent the meaning of a document, rather than the profile of a category. TF-IDF is a way to weight features that appear in a document, in order to point out the most relevant, and it is usually applied in conjunction with another weighting technique for the profile (Rocchio's or Widrow-Hoff, for instance). Still, this technique could also be applied to profile weighting.

The definition of this measure relies on statistical properties of the distribution of the features in the collection of documents. The intuition is the following: the importance of a meaning is proportional to the number of times it occurs in the document and inversely proportional to the number of times it occurs anywhere else. To make an example, if a document uses a rare word a lot, that word is likely to have a role in the meaning of a document. If it uses a frequent word a lot this does not give us much information, while if it uses a frequent words only a few times it means that word has nothing to do with the topic treated in the document.

To formally define TF-IDF, we will use the following notation:

$$DF(F_i)$$

to indicate the number of documents belonging to a collection D in which the feature F_i appears.

We will then define $IDF(F_i)$, the inverse of the frequency, as follows:

$$IDF(F_i) \triangleq \log \frac{|D|}{DF(F_i)}.$$

We will also define $TF(F_i, d)$ as the number of occurrences of the feature F_i inside document d.

The weight will then be evaluated by the following formula:

$$TF\text{-}IDF(F_i, d) = TF(F_i, d) \cdot IDF(F_i).$$

Documents are represented as vectors made of the same features as the profile, whose weight is given by the above formula.

A mean to obtain profile weights from documents' is as follows: the profile of a category P_C will be a vector:

$$P_C = \sum_{\vec{d} \in C} \vec{d}.$$

In our special case, TF-IDF would suffer from some peculiarities of the domain. Documents are represented by commentaries, so their length (and hence the number of occurrences of a term) is not significant. Estimations could be way wrong. Also, if it is not terms that are indexed, frequencies are even lower, thus making any frequency data unreliable. For these reason, TF-IDF is not expected to produce better results than other techniques, in this case.

5.2 Rocchio's weighting technique

Another popular way of weighting features is called Rocchio algorithm (a description of which can be found e. g. in [11; 10]).

The weight associated to each feature is greater as greater is the percentage of documents which have that feature and that belong to the category. Conversely, the weight will be smaller the bigger is the percentage of documents which have the feature but which do not belong to the category.

The weight associated to the j-th feature of the profile is assigned by the following formula:

$$w_j = \alpha w_{1j} + \beta \frac{1}{|C|} \sum_{d \in C} x_{dj} - \gamma \frac{1}{|C|} \sum_{d \in \overline{C}} x_{dj}, \tag{4}$$

where α, β and γ are parameters and can have different values: in many cases, standard values are used, such as 0, 26 and 4 respectively (as done by Lewis in [10]). In the formula, x_{dj} is the weight of the j-th feature for the d-th document. The information used can by e. g. the one provided by TF-IDF or could be a simpler value, such as 0 if the feature is absent, 1 if present.

This technique derives from studies on *relevance feedback*, where the profile is supposed to change according to some form of feedback. Rocchio himself does not guarantee that the measure will perform reasonably in other contexts, but it is nevertheless one of the most exploited.

The use of the α addendum in the formula was to evaluate the contribution of the weight vector of the previous step when adjustments upon user's behavior are considered [18].

5.3 N-Doc weighting technique

Other than the typical weighing techniques, we have defined some of our own, for the purpose of identifying characteristics of the features not necessarily bond to a classification task. For this reason, several different values can be interesting to us, in order to have a spectrum on which to evaluate the behavior of the system.

One of the techniques we defined is called N-Doc, and it is trivial. It simply counts the number of documents belonging to a category which have the feature. It is a sort of simple TF-IDF, where only the frequency part is present (not the inverse frequency).

Formally, the weight w_F of the feature F is defined as follows:

$$w_F = |\{e = \langle F_e, L_C \rangle \in E_C | F \sqsupseteq F_e\}|.$$

Intuitively, the highest number of examples have the feature, the more that feature is "popular" (and hence important) for the category the documents belong to.

It is easy to criticize this approach by pointing out that very common (and thus irrelevant) features distributed all over the collection would have high values by this measure. While this is definitely true for a system that would weight all the features in the collection, because we removed the features that were considered ambiguous (and a very common feature would be so), we can expect such features not be present in the profile. It is true, however, that for its naive straightforwardness, this measure would not be a good choice for a classifier.

5.4 Doc-Uniq weighting technique

This weighting technique assumes that the best features for a category are the features that are necessary for the highest number of documents. By the way the state evolved, we know that if a feature is necessary for some example, then it is the only reason that was found so far for which the document should belong to the category. If a feature is the only reason at all for many documents, it must be a very strong feature for the meaning of the category.

The weight assigned to each feature is:

$$w_F = |NEC_e(F, C)|.$$

This weighting technique only works for structured features, since no term would probably be necessary. Also, because the weight is evaluated on examples, the necessity of an examples makes it a good measure, really, only when the features have the same depth as the documents.

This measure, has a serious drawback. While it is true that necessary features are the only evidence of many documents in a category, nobody ensures that such evidence is good evidence. Because the feature is necessary it will not be eliminated, and it will therefore possibly propagate to a point where it is too specific.

5.5 Probabilistic weighting technique (inf-pos)

The more effective original weighting technique that we experienced in the tests we made is based on probabilistic assumptions.

The use of weighting measures which try and estimate a probabilistic distribution are very common in literature (see e. g. [17]), even though the method we propose is different.

We weight feature by assigning it an estimation of the probability that an example which has that feature belongs to the category. We called this measure *inf-pos*, that is, the positive information that the presence of a feature gives us when it comes to the decision of whether or not to assign a document to a category.

Formally, we define the weight w_F of a feature F as the probability $p(C|F)$, where C is the category to whose profile F belongs. The probability is the probability that a document belongs to C given that it has F. We estimate this probability over the training set:

$$p(C|F) = \frac{|\{e = \langle F_e, L_C\rangle \in E_C | F \sqsupseteq F_e \wedge F \in P_C\}|}{|\{e = \langle F_e, L_C\rangle \in E | F \sqsupseteq F_e\}|}.$$

that is, the ratio between the number of examples subsumed by F and belonging to C and the number of examples in the whole collection subsumed by F.

As it is evident, this measure works with the same principle as Rocchio's: the more examples have the feature F and belongs to the category C, the greater the weight is, the more examples have F but do not belong to the category, the lower it is. It is a direct proportionality, however, unlike Rocchio's.

This measuring scheme has multiple advantages: it is independent from the type of objects that are weighted, so that it is suitable for all types of features. The weights are probabilities: this allows an analytic estimation without a need to compare the weight of a feature with others'. We know that the closer the weight is to 1, the higher the probability that the feature is a very good descriptive one for the category is, while a feature whose weight is close to 0 is unlikely to be a good concept for the meaning of the category.

There is another advantage in having probabilities as weights: it is possible to combine them into functions whose analytic meaning can remain clear: there is no need (in principle) for a trial and error process to fine tune a matching function, for example. Also, the probability estimation is independent from the model used later to match documents against the profile, and it is therefore a good general measure (though probably not the best overall).

One of the disadvantages of this measure is that it does not take into account the frequency of a feature at all. Very rare features can have misleading estimations due to their rarity.

5.6 Negative evidence weighting technique

The goal of the measures described so far was the one to give an indication of feature relevance in the profile of a category it belongs to. We defined as "negative evidence" a measure that considers the importance of a feature relatively to the categories to the profile of which it does not belong.

We assign to each feature $n-1$ weights, where n is the number of categories. Weights are evaluated in a fashion similar to the one seen for inf-pos. The probability estimated, however, is $p(\neg C|F)$, that is, the probability that the example does not belong to the category C, given that it has feature F.

Formally, we say that:

$$p(\neg C|F) = \frac{|\{e = \langle F_e, L_C \rangle \notin E_C | F \sqsupseteq F_e \wedge F \notin P_C\}|}{|\{e = \langle F_e, L_C \rangle \in E | F \sqsupseteq F_e\}|}.$$

where E is the set of all training examples.

It is very likely that the weights calculated as above are very high: given the measure definition, they will often be close to 1. This is due to the selectivity of the features, which comes from the algorithm scheme as described in section 3.7.3.

This measure should be used in conjunction with other types of measures that evaluate positive evidence as well.

5.7 Selected measures

In the generality if the instantiation, the choice to the weighting measure should be left. Of the measure that we just described, we chose to use (for testing purposes) the measure we called "inf-pos", due to its generality and independence to the feature structure and matching model. We have also experienced that, for classification purposes, the additional information provided by a measure such as "inf-neg" has proven to be valuable, increasing performance.

6 Profile consolidation techniques

Once the training phase is over, it is possible to perform some post-processing over the features, in order to improve the profiles. This phase can be added to the feature weighting technique we just discussed.

In this section, we will also discuss some improvements made to the learning algorithm, which were not part of the model because of their relationship with some of the instantiation parameter, but that can be generalized to be considered a general mean to improve performance.

6.1 Profile processing

In literature, we find techniques such as term space reduction, which "manipulate" the profiles in a different stage than the training one.

There are many reasons why it is worth to process the profiles after the termination of the learning algorithm rather than during the algorithm itself, but the main reason is that the algorithm is set so as to learn continuously, in a virtually unlimited learning. This causes a continuous variation in the profiles, with the specification and generalization of features, which does not necessary lead to a convergence, and leaves the system slightly unstable. We can picture this situation in an empirical, yet intuitive way comparing the profile to the surface of an n-dimensional (where n is the dimensionality of the feature space) structure. This structure, whose surface is made up of features, which represent points in the space, continuously evolves during the learning, changing its shape. An overview of this situation stripped down to a bi-dimensional space is shown on figure 7.

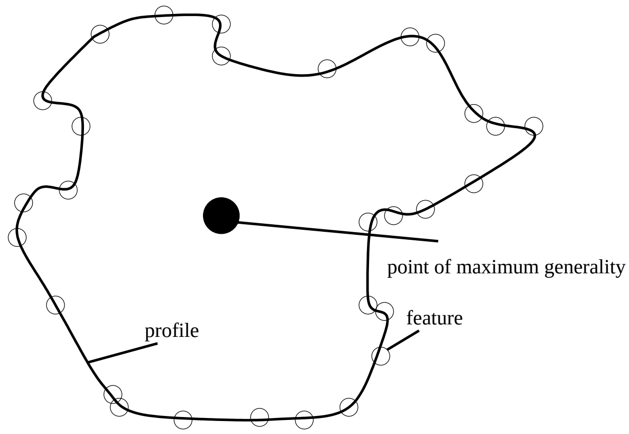

Figure 7: **representation of a profile as a surface in space. The profile features lie on the surface, and move away from the center of it as they get specialized. More specificity also means, intuitively, a more "rough" surface.**

We can say that the volume of the structure represents the features the system "ate" during the process. Even though they do not appear in the profile anymore, that does not mean that they are "forgotten". It just means that the system, forced by ambiguity, explored a more specific and precise concept. Because, however, virtually any feature would be ambiguous under certain circumstances, it could be worth to reconsider the features in the "volume" despite some degree of ambiguity. This way we would gain something in generality, and lose something in discrimination capacity: the system tends to keep itself in a "safe" state, behind a fairly specific horizon, in order to avoid ambiguity.

If it is true that many features in the inner volume could worth be reconsidered, it is also true that not all features in the profile are worth keeping. Because the learning process is never ending, at any time the system is interrupted, although coherent, the state will present "exploring" sides to it. If, again, we picture the profile as the surface of our structure, we can imagine that there are parts of the surface which are "edging out" trying to explore the opportunity to move towards that direction.

For this reason, it is worth to consider a "smoothing" phase, which should prune all asperities from the surface, or at least all the non significant ones.

We cannot say whether or not it exists a state in which the system would no longer learn, which could be represented as the configuration as a stable shape in the space. From experimentation, we have observed that when categories are fairly general (for instance, the main categories of Virgilio's catalog), the learning process has a slow slope: changes in the state are fewer as more examples are observed, but they are still fairly frequent after 20.000 documents, and with a slow decreasing attitude.

We have studied techniques to deal with these situations, and modify the profiles after the training process is finished, in order to retrieve some feature, and generality.

We can divide the techniques used in two separate parts:

- enriching of the profiles with extra features obtained from the ones available at the end of the training. This way, we try and gain back some generality, while still keeping the more specific features for a more detailed description. Ideally, one could trade some ambiguity for generality, and keep only the features "not too ambiguous";
- elimination of useless features, thus making the surface "smoother".

6.1.1 Profile enrichment

How is it possible to add features after the training process is finished, and therefore no more examples are examined?

To describe this procedure, we must consider what we imposed to the features in the description of a category. In particular, we must consider that we wanted a feature to be part of just one profile.

This requirement can be too strict in many cases, especially since every concept is ambiguous at some extent. A certain degree of ambiguity could be allowed, not to obtain too much of a specific description.

Post-generalization We applied a technique that we called *post-generalization* to the profiles obtained from the training process.

With this technique, we look for new feature from the features that are part of the profile, following a generalization step similar to the one applied during the training phase.

The main difference between this generalization and the one used in the learning algorithm can be summarized in three points:

- in the post-generalization no new examples are involved: new features are obtained from generalization of couples of features;
- the threshold of a feature is not considered during the generalization process: any generalization is allowed;
- features obtained via post-generalization are not checked for ambiguity as it used to be in the training process.

The second point is the one that effectively introduces the "desired" ambiguity in the description. Feature thresholds, that used to impose the formation of "safe" features, are not considered, and therefore ambiguous features are allowed to form.

Thinking of the profile as the aforementioned surface, this process can be seen as a motion from the surface to the inside, in order to reconsider the "volume" of internal and more general features. This is intuitively visualized in figure 8.

One can question whether this process is correct, and why should anyone bother specializing during the training phase if generalization is performed afterwards.

The search for specificity in the training phase is required by the imposed distinctiveness of the features that we want in the profile. The assumption that a correct description can be achieved unambiguously is fairly strong and could be impractical in some real case. It can therefore be worth it to partially release this requirement in order to obtain a more general description of concepts.

Obviously, we cannot assume that all post-generalized features are relevant to the meaning of the category. Many features were specified during the training process just because they were

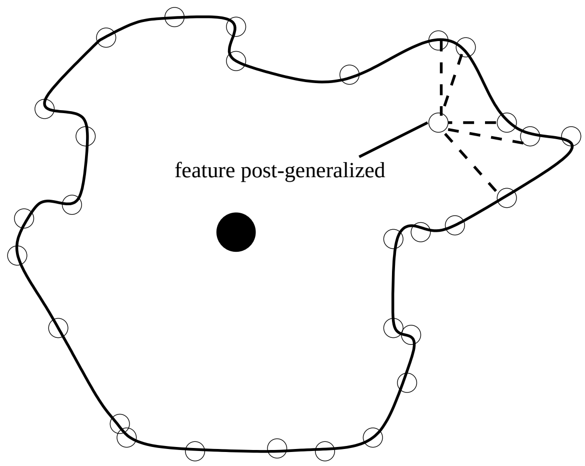

Figure 8: **a case of post-generalization: features belonging to the profile (on the surface) are generalized to regain a feature on the inner volume.**

"bad" features in their general form. We expect, however, that such features would obtain a low weight during the weighting process, and could therefore be identified.

A few examples can make this situation clearer. Suppose after the training process we have obtained the following features (depth 1 features) for the category labeled as "Sport". F_1 =`(SOCCER MILAN)`, F_2 =`(SOCCER JUVENTUS)`. It would be desirable to have the more general feature `(SOCCER)` in the profile, as well. By generalizing F_1 and F_2, this can be obtained as a feature F_{1+2}.

Consider F_{1+2}. The presence of the simple term `SOCCER` makes it definitely more ambiguous than F_1 or F_2, but it can be kept as a good description for the category (compatibly with the domain, of course). During the training process, it is very likely that such a feature was created and then specialized when an ambiguity was found, for instance for the creation, in a category such as "Video-games" of a feature such as `(SOCCER VIDEO-GAME)`.

The above example suggests us another thing to consider: if in the category "Video-game", with the post-generalization process, the feature `(SOCCER)` was obtained from more specific ones, we would have a misleading feature in the profile of the category. We expect, of course, this feature to have a low weight.

We can then make some observation about the profile composition and on the opportunity to eliminate from the profile the specific features that have been post-generalized. These features can be left in the profile if we want to obtain a high discriminative description. The presence of more specific features, in fact, reduces the uncertainty of the description.

In this case, of course, we must consider that not all features are independent.

6.1.2 Cleaning techniques

During the feature extraction process, some useless or discardable features form. Because the formation of features is unsupervised, there is no control over this fact, except for the ambiguity resolution. For the system, a way to eliminate such features is the check for ambiguity, which is deliberately not done in this context.

Looking again at our n-dimensional structure, such elements could be interpreted as "bumps" or asperities in the surface: in such points the system did not obtain a satisfactory generalization,

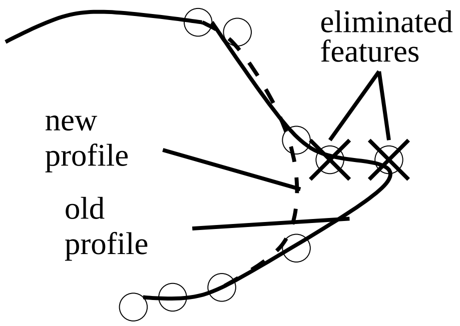

Figure 9: **an example of "smoothing" of the surface. Two irrelevant features on the surface are eliminated, making the profile "better".**

and then the surface does not appear smooth.[5] Expunging these features would make the surface smoother, and leave only significant features. An intuitive example of what happens is shown in figure 9.

Other undesirable features are the ones which we can consider improper. With this term we indicate the features which are part of the profile but which were generated by a very small number of examples (for example, 1). In the limit case of 1, in fact, the feature is a training example itself, and did not undergo any generalization at all.

If some of these features might fall in the "bad feature" case, as described above, some of them might not. Because of their specificity, the frequency of such features can be insignificant, and any estimation of its relevance misleading. Because these features appear to be very marginal in the training set, we would rather eliminate them, while still remaining sure that we do not lose much information.

Cleaning techniques have thus the goal to "clean" the set of features, and because it is done *after* the training process, it could be considered analogous to a term space reduction process. Because our features are not always terms, we have experimented a few cleaning techniques.

Minimal number of examples The first and more simple technique is the one to remove from the profile those feature which were generated by a number of examples smaller than a threshold k.

The best value for k depends on many factors, such as, obviously, the number of training examples. Also, k could be chosen so as to have a more general or vague descriptions: for consequence of the training process, features which subsume more examples are also very likely to be more general. Such a simple mean to reduce the feature space is very common in literature, e. g. in [11; 7].

For our purposes, a very special value of k is 1, since we observed how features generated by just one example are not very informative at all.

[5] We have absolutely no proof that a good generalization corresponds to a smooth surface. This is an intuitive vision, where right things are nicer (and also, we are assuming that smooth is nicer than rough). This is not uncommon: we think that a good program to solve a problem would be "elegant", and that a good model of reality is the "simplest" and nicer (and this is the cause of many improvements in physics, in the past). When things get complicated, we get the feel that they might be wrong (as we used to think during tests in math classes). The minimum description length principle (or Ockham's razor in other contexts) is just another example of such a philosophy, which we apply as well (we ask for short descriptions). In this context, we just have to imagine the "bad", rough and non smooth shape as the wrong profile, and the smooth one as the correct.

For classification techniques, as far as the value of k is "decent", profile cleaning will not affect performance significantly, since we only remove very rare features.

Minimal weight of a feature The second cleaning technique we describe is still a threshold-based technique. It is intended to remove those features which are not relevant as measured by the weighting function "inf-pos". In particular, we will remove all examples for which the value of inf-pos does not surpass a threshold which depends on the category.

For the threshold, we use a value that is proportional to the probability that a document belongs to a category ($p(C)$). Because the inf-pos weight of a feature is $p(C|F)$, if this is lower that $p(C)$, then F should not belong to the description of the category C, since it is less likely, for a document, to belong to C if it has F than if it does not.

Intuitively, this condition would also give us an idea of the chance of a feature of being ambiguous. If F is more common outside C than inside, then it is likely to be ambiguous.

This expunging technique is really helpful if applied after a post-generalization step. Because post-generalization acts "blindly", it is likely to cause the formation of meaningless or irrelevant features made up of very common (and hence not very meaningful) terms, such as, considering our test domain, "site" or "home page". Without an expunging technique to remove all the overly ambiguous features, it would not be convenient to perform a post-generalization step.

6.1.3 Using the described techniques

None of the exposed techniques formally depends on the others, and they can all be applied. The best results in the test phase were obtained by applying them all.

The most effective way of combining the techniques, is to apply them in a particular order, which is also the most intuitive. After the training process, the first step to be applied should be the post-generalization one. This way, as discussed elsewhere, we bring the system back to a more general, yet more ambiguous state.

Cleaning techniques should then be applied to try and tell the generality apart from ambiguity, if possible. As it is obvious, the cleaning technique based on the inf-pos weight must be applied after then weighting phase. As this phase is relatively time-consuming, it is worth to first apply the very simple cleaning step based on the number of examples generating a feature. After this is done, reducing considerably the number of features, the weights should be calculated, to finally proceed to the cleaning based on the weight.

6.2 Training boosting techniques

The proposed learning scheme learns from examples with no external information other than the tagger.

The supervised learning implemented has an unsupervised "core", which is constituted by steps 2 and 3 of the algorithm described in 3.7.3. It is therefore useful to apply some additional techniques as found in literature to make the job "easier" to the learning process.

6.2.1 Stop word removal

The technique known as *stop word removal* is very popular amongst text classifiers based on simple terms. This technique consists of the use of a list of "bad" terms, the elements of which are removed from the training examples before any training begins. Members of the list can be meaningless particles, such as "the", "as", "a" and the like, as well as very frequent words, or words meaningless to the domain. By removing *a priori* features that can be considered useless, the work of the training algorithm is made easier.

The stop word list is usually constructed by hand, which is obviously an expensive and error prone process. Also, if on one side it allows to add words that are meaningless only in the particular domain the system is going to be run, for the same reason it reduces its reusability. Finally, for as a human can give much information about meaningless terms, there will always be terms she leaves out but that should be included in the list.

In our instantiation, much of the work usually done by the stop word removal step is actually performed by the tagger, thanks to its identification of the tags, which allows us to leave out all conjunctions, prepositions and the like.

Still, most of the nouns which are irrelevant to the domain would make it through the tagger. In particular, it would be desirable to discard terms such as "site" *before* the training process begins, so as to make it more effective and easier. The more the definition of the predicate "$Captures$" is strict, the more desirable the stop words removal is, since the already high frequency of a term would be amplified.

We have developed a mechanism that allows the system to identify "dangerous" terms and make sure they do not form features on their own. Stopped terms will not be allowed to form generalization when no "good" term is present, so that their influence in the profile will be reduced, and, most important, they will form fewer and fewer times during the training phase, leaving space for other (possibly relevant) features.

The system we developed is adaptive, and does not require external information nor an examination of all documents beforehand.

Every time a feature is generalized to the point it is composed of just one term we keep track of the fact. If the feature turns out to be ambiguous, at some point, we increment a counter associated to the term in an associative table, so as to know, for each term, how many times it had to be specified. By the algorithm definition, meaniningless, desirably stopped, terms as "site" will cause many ambiguities, and will tend to form features made up of just that term many times. Periodically (typically every example) the terms are ranked according to the umber of times they had to be specified, and the "top scorers" are considered bad, and hence stopped.

Because a stopped term cannot form features anymore, sooner or later some other term will reach that number of specialization required. For this reason, every time a generalization would cause the formation of a feature with that single term, and is therefore considered invalid, we add 0.1 to the ambiguities counter of the term. Even if an ambiguity cannot be revealed, when a bad term appears very frequently, we would rather keep stopping it. Although this is the technique we used, improvements can be made: the most obvious of which is to consider the ambiguity probability, estimating it from past experiences.

The proposed method has an interesting peculiarity: it allows to identify stop words automatically without examining all examples beforehand. Also, the *order* in which examples arrive matters, which is not necessarily a bad thing.

The first of these two peculiarities makes this technique applicable to a system which receives and learns from examples in real time, that is, interactively. It does not know what example will be shown to him last, so it has to be adaptive. The semi-incremental training algorithm we just proposed, therefore, could be aided by this technique in an on-line form of learning, possibly for a life long learning scheme.

The second of these two characteristics is more subtle, and suggests that, somehow, the *ambiguity of a term can depend on time.* Usually, in fact, it happens that some event causes a big influx of documents in which terms have a different meaning than they usually have.

Alternatively, there can be many references to a particular situation or event which alters the usual meaning of words. Yes, we do have an example of such situations, to make things clearer. Consider the word "cow". Usually, we would place this term in the description of a term such as "Farms", or "Environment", or "Domestic animals" or the like. Still, there was an abnormal rush of such a term in medicine oriented documents when the "mad cow" disease was hitting the news (other examples, as well as the whole "media rush" idea, are described by Yang et al. in [33]). Similarly, the same term was abnormally frequent in documents talking about "Politics" or "Economics" at the time of the strikes and manifestations of farmers about taxes over milk, when in Italy a cow had its month of glory for being the symbol of the protest.[6] While usually the term "cow" would not be assigned to "Politics" or "Medicine", it might have had in that period. This means that the term was ambiguous at that time, while it was not earlier. What happens once the news rush is over? Cows are back to the farm: the term will start appearing again only in a farming environment. With time, it will drop out of the stop list, to become again a good member of just the "Nature" or whatever category.

On a side, therefore, we will have the elimination of words that, in this time phase, are ambiguous, while on the other the system will be able to adapt itself to changes in the meaning of words. At any time, therefore, descriptions will be good descriptions for the temporal frame.

6.2.2 Past example ordering

In the step 3 of the algorithm described in 3.7.3, past examples are "shown" to the new feature, in order to identify previously missed similarities amongst documents. Because the steps 2 and 3 of the algorithm realize a non supervised training on the category examples, it is important the *order* in which examples are examined.

To re-examine the examples always in the same order for every feature means grant the first example higher chances to form features it is subsumed from, while fewer and fewer chances are granted to the following examples. This is because generalization causes some information loss, and the information, at any step, is the intersection of all examples examined so far. Examples will definitely have a void contribute to the generalization after the first few generalizations are made.

This is one of the reasons why in non supervised learning algorithms examples are usually examined in random order. Since the algorithms usually take more than one epoch to learn, at any epoch the order of the examples must be changed, so that all examples have the same chance to be on top of the list.

In the case of the learning algorithm we defined, we can make further observations, so as to define an ordering which will perform better than the random one. Because old examples are examined by generalizing every time, at every step information is lost. The fewer information is lost the better, since the more we lose, the more it is likely that there was valuable information in what we discarded. Also, randomizing the order would give frequent and meaningless features an extra change to show up.

The solution we propose is to match examples against the new feature as ordered by the value of $\mathcal{G}$ applied to the generalization of the example and the feature. Every example is matched against the feature, and the one whose generalization causes the smallest loss of information is chosen first. This way, at the first step we have as little a loss of information as possible, and

[6] In 1998, italian milk producers protested against fines that were imposed to them by the government because of the extra production of milk. The protest was brought on by saturating and blocking highways and by bringing this cow, the "mucca Ercolina", all the way to Rome with a slow trip as a symbol of the movement. It has been one of the main issues in the news for about a month.

we heuristically assume that the same stands for the other steps. This is not entirely true, since after the first generalization step the optimum ordering might change, but since it is impractical to do a complete generalization and sorting step for every example, this is a good trade-off.

The problem of very frequent features is not completely solved, this way, but we still achieve some local minimization of loss of information, thus giving an extra chance to "good" features to form.

7 Classification

Automatic classification[7] is the activity of assigning one or more category to a document, depending on the similarity between the concepts expressed in the documents and the ones making up the category meaning.

This activity, if it has always been matter of research, has gained much importance in the past few years, as digital documents became the most frequent medium even for non structured data, and with the birth and impressive growth of the *World Wide Web*.[8] The increasing number of digital documents, as well as their availability, gave anyone a chance to reach more information than a few years ago was imaginable.

The problem is in organizing, fetching or retrieving such a huge amount of information. It is like for a customer who enters for the first time in a huge library. The information she is looking for will most likely be available, but where is she going to find it? Without an index, a catalog or (better, to be honest...) a library guru, all the information available would be completely useless.

There are several ways to retrieve the desired information, amongst which the "catalog" form has a number of advantages. First of all, the user often does not have a precise idea of the topic to look for, and searching for very general topics can cause the retrieval of so many documents that the search over the reduced documents would still be very impractical.

The catalog can give many hints to direct the search more effectively and to organize the user's ideas. The catalog organization, also, allows a general search as well, like a search engine. Search could be done inside a category, so as to disambiguate search keywords and retrieve a considerably smaller number of documents.

The manual construction of a catalog is undesirable, although most of the catalog organized search engines are still built manually. This solution, in fact, requires the employment of many "experts" to classify with a tedious and unattractive work many documents. The number of documents to be classifies is too large for manual intervention to be effective, and allocating the work to many contributors can cause inconsistencies. Experimentally, it is acknowledged that even a single person would not necessarily be consistent in classifying.

As an extra difficulty, especially for Web pages classification, we must consider that the domain changes constantly and significantly. While a missing link can be detected automatically by a spider, a significant change in its content, which would make it migrate to another category, is much harder to detect automatically. As soon as an automatic way to determine the shift in

[7] In this context, "classification" and "categorization" are used interchangeably.

[8] It seems like the Web phenomenon gave a significant boost to nearly all kinds of research. Almost no research became obsolete because of the Web, while interest seems to be born again where it was dropped with the Internet. We believe that this is because the Web increased significantly the "contact" to low end users. While, a few years ago, a typical user only came in touch with what he wanted, nowadays data are almost streamed to the users. For the hetereogenity of the data, nearly all kinds of activities and techniques become useful not only to the scientific community (as many time, although sadly, happens) but to all users, increasing benefits, and thus encouraging research.

a document meaning is developed, manual classification would be obsolete. The speed at which the domain changes is therefore an issue, and one that brings manual classification further down.

Because the extraction of category meaning is very close to classification, and because we were inspired with our system from classification algorithms as well, in this work we explore the opportunity to use the descriptions we obtain by the instantiation we described elsewhere to perform classification of new documents. We must consider that a correct classification was not amongst the goals of this research. Obtaining a better classification is more like a medium term result, whereas in this work our main goal is to produce good descriptions. We believe that this constitutes a brick towards the construction of a possibly better future classifier, but we also admit that much work has to be done in order to develop a classification method that would more precisely take advantage of the information extracted by our system.

Because we have no objective way of evaluating our work, however, classification is a way to do an indirect evaluation of the quality of the descriptions obtained. As for expected results, one should be pessimistic, as research done on classifiers have shown quite disappointing results whenever features more complicated than simple terms were deployed.

While we have no test of a standard classifier performance over the same collection, we believe that the instantiation we gave based on single terms could be considered as a good match, as it relies on basically the same assumptions and theories as classifiers such as Rocchio's.

7.1 Traditional techniques

Classifiers are composed of different modules. Documents are first indexed, by choosing the features, later a profile is constructed by combining the documents into a profile of weighted features, and finally a matching function is deployed to, together with a thresholding strategy, allow the classification of new examples. Relatively to the algorithm we described so far, we can say we defined a way to index documents and create the profiles of weighted features.

Traditional models, over these modules, differ from the proposed model mainly in the choice and weighting of features.

In the present section we will describe classification techniques that can be used in our case as well.

One of the main differences with our model is the document representation. Usually, a representation by content is used, so as to obtain more frequency related informations, while we chose a representation by context.

In a standard technique, one would consider all terms in the documents, or optionally apply a term space reduction technique, which can be based on several factors, all usually estimated statistically. One exception is the stop list, which is entered manually. The *information gain* technique, on the other side, evaluates a measure saying how "informative" a term is, and cuts off the bottom part of the term list, as ordered by that measure. Reduction can be impressive: a tenfold reduction can not only guarantee the same performance, but sometimes it might make the system perform even better.

Once terms are selected, documents are represented as vectors of weights associated to terms. Vector representation makes it easy to evaluate the *similarity* between two documents. Intuition suggests the most widely used similarity measure: the *cosine correlation.* The similarity between two documents is evaluated by calculating the cosine of the angle between the two vectors representing the documents in the document space. Let v_1 and v_2 be two vectors, the similarity

S between the two is calculated by the formula:

$$S(v_1, v_2) = \frac{\sum_{i=1}^{n} v_{1i} v_{2i}}{\sqrt{\sum_{i=1}^{n} v_{1i}^2} \cdot \sqrt{\sum_{i=1}^{n} v_{2i}^2}}. \tag{5}$$

7.1.1 Document-centered systems

Document-centered classifiers do not build profiles to represent a category meaning, and simply store the training examples. In a way, we can say that, for each category, the set of examples belonging to it represents it, and it is therefore a sort of profile for it.

In these systems there is no proper abstraction step: the learning phase is as simple as storing examples in an appropriate format. For these methods, learning means memorizing.

Confrontation with this kind of systems can thus be done only at the classification level.

To classify a document, it is matched against the examples examined during the training process: by a similarity measure such as (5), the document is assigned to the category to which the most similar examples belong.

The most popular system of this kind is called *k Nearest Neighbors* (kNN), as it considers the categories to which the k examples closest to the document being examined belong.

The value of k is one of the system parameters, and determines the size of the interval, centered at the document being classified, in which documents should be considered "close", in order to mime their classification. By posing k to 1, as a result we would have a system that classifies a document according to the categories to which the closes training example examined belongs.

Even though kNN does not construct profiles nor extract any explicit information from the training examples, it is possible to set some of our model parameters so as to obtain a behavior identical to kNN's. This stands as a verification of the model's generality.

The depth of features must be identical to the one used to represent documents. Because a profile is a set of features, and kNN "profiles" are set of documents, we want a feature to be identical to a document.

To avoid any kind of abstraction of information, we "fiddle" with the initial threshold. Because generalization (and thus abstraction) has an upper limit in the value of the threshold, we will have to make sure no generalization is ever allowed. This is not difficult, as it can easily be done by putting the initial threshold Θ to the ideal value of $+\infty$. This means that since no generalization will ever be possible, every time a new example is matched against the profile, all examples will have to be added to the profile, no matter what the definition of the predicate "*Captures*" is.[9] No ambiguity check is wanted, and the ambiguity predicate can then be defined as constantly false, but even if performed it would not have any effect, since documents are so specific that ambiguity is unlikely. When a document (and then a feature) appears identically in the profiles of two categories, no ambiguity is detected, since the same document caused its formation.

Omitting, then, any kind of weighting or expuntion, we will obtain a set of "features" in which each feature represents exactly (and identically) a document.

To finally get to an exactly equivalent classifier, it is sufficient to define an appropriate matching function, which considers all profiles of all categories to determine a document's classification. For each feature in the profile, similarity via the cosine correlation can be used to determine how close the examined document is.

---

[9] Except, of course, the trivial (and useless) case in which the predicate has value $\top$.

Other than the cosine correlation, which is the one usually used in kNN, and which therefore grants the same classification performance, we could experiment different measures. As a trivial example, we could use, instead of the similarity measure, the function $\mathcal{G}$ applied to the generalization of the document with the feature. This way we would have a match that would try and save as much of the original structure as possible, differently from what the cosine correlation does. It is interesting to observe how a document close by the cosine correlation definition would not necessary be as close by the $\mathcal{G}$, while if a document is close to another by means of $\mathcal{G}$, then it necessarily is by the cosine correlation as well. Although less stable,[10] $\mathcal{G}$ applied to the generalization of two documents would therefore be a more strict measure.

7.1.2 Category-centered systems

Category-centered systems are more similar to our model. Even though the model has been proven to be able to "mime" kNN-like systems, it would not make much sense in such a configuration, since it would basically do nothing but store examples in a perverted fashion.

Although its ability to adapt itself to completely different learning models is exotic, it is more interesting to compare the system behavior to the ones of systems that build proper category profiles, trying to abstract information.

Whether or not a document belongs to a category is a decision which is taken by a match of the document against the profile. The main deal with these systems is to define a measure to evaluate the similarity between the document and the profile of the category: the higher the value is, the more likely it will be that the document belongs to the category.

7.1.3 Parametric and non-parametric systems

We say that a system is *parametric* if it uses the training set of examples to estimate a probability distribution.

The assumption is that the distribution observed in the training set is a significant sample of all the domain one, and that documents will therefore follow that rule. These systems will then build an evaluation of the probability $p(C_i|d)$ that a document d belongs to the category C_i.

A probability estimation gives us a more general information than a simple similarity measure, since we can use that information in an analytic way, or for purposes other than classification. While the values of a similarity measure can be nonlinear and hence hard to interpret, we can always interpret the probability that a document belongs to a category is p.

The main drawback with this kind of system is that should the distribution in the training set not match the one of the documents that the system will have to classify, the algorithm will be affected by a systematic error. This is especially true when the system is fed with documents to classify that arrive in real time on a stream: the document distribution will be affected by time-depending changes which can be significant.

Non-parametric systems are systems in which such assumption is not made. They simply look for a good classification scheme, and thus the information they give is only good for classification purposes.

[10] $\mathcal{G}$ would be less stable because it takes discrete values.

7.1.4 Batch and incremental systems

A transversal distinction of classification algorithms, or, more generally, of learning algorithms is the batch vs. incremental dualism.

Batch systems work monolithically over all the set of examples, exploiting them in the way they want, eventually scanning them all multiple times. All the training set must therefore be available when the training begins. Once computation is over, the state of the system is stable and non-elastic: to make an adjustment, a whole recomputation is required.

These systems have usually better performance over incremental systems, since they can do more operations over the examples. For instance, a multiple epoch training could be deployed, or a wrapper-like self evaluation scheme to fine tune the training and avoid over-fitting the data.

These systems are well suited when a skill or a classification ability has to be learned once for all. For this reason, these are valuable schemes when we know that an optimal solution can be found, so that once the training finds the solution, it can be frozen and exploited. An example of such a scheme could be decision tree learning deployed when an exact solution is known to exist (e. g. in the card game Eleusis [25]).

Incremental systems, on the other hand, examine training examples one by one, discarding them after each example has applied the wanted modifications to the state of the system. In this sense, the concept of "collection" of training examples is simply due to the fact that the training process is stopped when no more examples are available. Still, the system is open to new examples, should any be available in the future.

This is very suitable for situation in which training is done in real time, as examples arrive, since there is no time to wait for a collection of examples to be available at once. Its lack of request of great storage makes these systems suitable for reactive structures such as neural networks. Also, they are suitable when we know that exact solution is unachievable: in this case, we would rather have a system that keeps learning approaching a good solution for the best it can, without eventually over-fitting the data, and adapting to peculiarities the domain can have, in order to achieve better results in the subclasses of the domain of interest. In this case, obviously, some form of convergence should make sure the system is not totally unstable.

7.1.5 Linear and non-linear systems

The distinction between linear and non-linear systems is done by means of the matching function.

Simply enough, a classifier is said to be linear if it uses a linear matching function, such as the scalar product of two vectors, while it is said to be non-linear when the function used is non-linear, like the cosine correlation.

Usually, non-linear systems have a better performance due to the complexity and richness of the matching function, but some form of probability estimation, for instance, can be done linearly.

7.1.6 Rocchio classifier

The Rocchio classification algorithm is one of the most used. This method was originally conceived to try and take advantage of "relevance feedback", but it has later been deployed in information retrieval tasks, and finally for automatic classification.

Every category is represented by a profile in the form of a feature vector, weighted by the measure (4).

To classify a document, the similarity between the vector representing the document and the profile of the category is evaluated. Since the profile represents a vector in the document space, the similarity measure will establish how close the document is to the *prototype* of document represented by the profile.

This technique has its major drawbacks in the representation of the meaning of the category as a prototype of a document. When the meaning of a category is very general or scattered, it is hard to represent it by a vector in the document space. The more general the meaning is, the larger the "circle" around this prototype of document that belong to the category will be, and thus the less precise the description is. In the case of a "scattered" meaning, that is of a category which is semantically subdivided into sub-categories, the situation will be even worse. Documents will tend to form clusters representing each one of the sub-categories, and the profile, a sort of a centroid vector, will be in the middle of all clusters, in a region possibly not covered by any cluster. As classification could still be effective, the profile will still not be a good description of the category meaning.

Consider the category "Sport". This category can be logically subdivided into "clusters" of documents talking about a specific sport, such as tennis, soccer, basketball and so on. Documents will tend to cluster with documents that talk about the same sport, while the profile, abstracting, will be the prototype of a document generally talking about every single sport, being pretty far from a typical document belonging to the category. Classification could still be effective, in this case, since it is unlikely that a document would be close to the profile while falling out of each cluster. For other categories, things could be worse.

7.1.7 Bayesian classifiers

Another very popular classifier is the *Bayesian classifier*. Bayesian classifiers, instead of defining a similarity measure, try and estimate the probability of $p(C|d)$ that a document belong to a category. Such a probability is estimated from the training set, making heavy and approximative assumptions.

The probability estimation is based on the Bayes theorem, to which this classifier due its name:

$$p(C|d) = \frac{p(d|C)p(C)}{p(d)}.$$

Assumptions are made in order to estimate the probability $p(d|C)$ from the training example. Since a document d is represented as a feature vector $\langle F_1, \dots, F_n \rangle$, the assumption of independence between features is made (see e. g. [19]), in order to be able to assert that:

$$p(d|C) = \prod_{j=1}^{n} p(F_j|C).$$

Each one of the $p(F_j|C)$ can now be easily estimated by looking at the training examples.

An immediate improvement to this scheme can be done by adding some form of negative evidence, thus making probability decrease when a document does not have usual features for a category:

$$p(C|d) = p(C) \prod_i \left(\frac{p(F_i|C)p(F_i|d)}{p(F_i)} + \frac{p(\neg F_i|C)p(\neg F_i|d)}{p(\neg F_i)} \right),$$

where $p(F_i)$ is the probability that a randomly selected document has feature F_i, and $p(F_i|C)$ is the probability that the feature F_i is in a randomly selected document belonging to the category

C. $p(F_i|d)$ is the probability that the document d has feature F_i, and is usually either 0 or 1, but more sophisticated measures are possible. The product is done over all features in the profile of the category.

Such an improved estimation considers more information for each document, since it does not only consider the document features, but also the profile features that are not present in the document.

On the other hand, using this function makes the system no longer a parametric one: despite the notation (which is present in most of the literature, as in [17]), the estimation is no longer a probability, as its value range is not in the $[0, 1]$ real interval anymore.

7.1.8 Other techniques

Many other techniques have been studied and presented, and many more could be invented. Some of these techniques come from a different study area, while other have been defined ad hoc for classification purposes. As an unsupported statement, we will say here that it seems like basically every common sense idea on classifying gives at least decent results.[11] Obviously, while more theoretically founded classifiers reach the state of the art performance, all others remain a step behind.

Many classifiers are based upon rules. Rules can be automatically generated, trying to infer from the training set, as with a decision tree learning scheme, or they could be manually generated by experts of the domain.

Some classifier had been defined over techniques that come from the field of *neural networks*, such as the one described by Ruiz and Srinivasan in [22]. These techniques are profile-based: they use learning models such as the *perceptron* one to incrementally construct the category profile.

One of the most used classifier of this kind is based on the *Widrow-Hoff* algorithm, also called the *minimum square* algorithm. It is a purely incremental method with which the profile of each category is defined as a weight vector w. At every step, the vector, which is initialized to $\vec{0}$, is updated according to the following rule (which refers to the j-th component):

$$w_{i+1,j} = w_{i,j} - 2\eta(w_i \cdot x_i - y_i)x_{i,j},$$

where y_i represents a label associated to the document represented by the vector x. The label is 1 if the document belongs to the i-th category, 0 otherwise.

The parameter η, which must be greater than zero, is called *learning rate*, and affects the change that a document will cause over the previously constructed profile.

This method is equivalent to a gradient descent, since the term $2(w \cdot x - y)x$ is the gradient, relatively to w, of the squared error, defined as:

$$(w \cdot x - y)^2.$$

Updating the profiles corresponds to a local movement along the direction in which the error decreases faster, so as to minimize the mistakes of the classifier.

While the more natural solution would be the use, for classification purposes, of the last profile as updated by the mentioned rule, experiments have shown that a better choice is the

[11] This is supported by the results we obtained in classification, which are close to the ones achieved by "guaranteed" classifiers such as Rocchio.

average of all profiles over the training examples, so that the final weight vector w will be:

$$w = \frac{1}{n+1} \sum_{i=1}^{n+1} w_i.$$

The reason why this vector should have a better performance is that it reduces the risk of over-fitting, and makes the system more stable.

7.2 Using the profiles for classification

Because we want to perform classification, we must choose a classification scheme from the one described above or an "original" one in order to obtain a classification rule from the description we have built during the training process.

The choice to experiment different techniques is justified by the difference of the training scheme and of the descriptions obtained, which justifies an effort to use their peculiarities.

One of the main differences is the profile structure as opposed to the document structure, as well as, of course, the use of features more sophisticated than single terms. This second fact makes it possible to take into account certain properties of features that were already identified by the researches upon classic classifiers, but that could not be exploited due to the choice of the feature structure.

7.2.1 Feature dependence

The feature structure allows us to evaluate the relation between a pair of features using only syntactic tools.

In Bayesian classifiers, the main and more approximative assumption is that all features are independent. Such assumption is due to difficulties in conditional probability estimation, which would make the training set dimension explode exponentially in order to have a significant sample.

Many solutions have been proposed to avoid such an uncomfortable assumption, and to propose a better model of reality, yet they all made calculations impractical without guaranteeing any improvement.

Correlation estimation techniques, which consider the frequency of a pair of terms, for example, risk to relate two terms that are actually independent despite their tendency to co-occur, which could be wrongly estimated if the sample is too small.

An effective, yet impractical solution, is the one to manually insert a dependency list into the system. This way, it would be possible to effectively give the system knowledge about unrelated concepts, but it is non-trivial, even for a domain expert, to build such a list coherently, nor the dimension of the required list makes it an easy task.

In our case, the feature structure allows us to evaluate some form of dependence between feature, coherently with the assumptions we made, by the use of the subsumption relationship.

When we consider the set of features a document has, we can discard all subsuming feature, taking only the more specific into account. If a feature subsumes another feature it cannot be considered by any means independent, since its occurrence is implied by the more specific's. We can easily eliminate, this way, the features which are macroscopically violating the independence assumption.

A further consideration can be made, to discard more features which violate the assumption. Suppose we have two features $F_1 =$`(A B)` and $F_2 =$`(B C)`. If a document is subsumed by F_1, its chance to be subsumed by F_2 as well is affected. The probability that it is subsumed by F_2 given that it was by F_1 is the probability to contain the term `C` in a noun phrase given that it already contains `B`. This probability is obviously greater than the probability to contain both terms in the same noun phrase without any additional information.

An independence estimation defined as such relies on purely syntactical features, and can thus be easily exploited.

If we can simply eliminate all subsuming features from the document representation, however, what should we do about features that are dependent, but not in a subsumption relationship? We could keep a pair of dependent features by lowering their weight so that their impact if dampened, but a better solution is to keep only one of the pair. Because the conditional probability is higher than the probability that both features appear in the document, but it is lower than the probability of either one of the two features, we underestimated the conditional probability by the probability of the more likely feature between the two.

7.2.2 Positive evidence

To perform classification, we decided to only consider the features in the document, rather than the features that the document does not have as well.

This choice is based on the consideration that since the categories we considered have a fairly general meaning, documents will tend to cluster inside of them. By putting a document down because it does not have features that describe another cluster of documents would excessively interfere with the correct classification of documents belonging to the least populated clusters.

There is another reason for which to base classification only upon features on the document side. While features made up of just one term have a broader punctual syntactical diffusion (they cover most of the document), on the other side their semantic ability to cover the meaning is opposite, since their meaning identification is smeared.

More structured features have the opposite characteristic. They represent only part of the syntactical form of the document, by leaving out a considerable part, yet their meaning identification is more punctual.

For this reason, we prefer to take into account the features that are part of the document rather than the ones that are not, whose meaning, being the inverse of a very detailed concept, is imprecise and smeared just like a term's one.

7.3 Numerical/statistical approach to classification

One of the main techniques for classification is the definition of a predicate which, given a profile and a document, determines whether or not the document belongs to the category.

For this purpose, it is usually deployed a function that is greater as there is more correlation between the document and the profile, so that a thresholding strategy can be defined to obtained a decisional predicate. Because the assignment of a document to a category is an independent event, a document can be classified to any number of categories or to none of them.

To apply this kind of classification scheme, we must therefore establish the similarity measure as well as the thresholding strategy.

7.3.1 Similarity measure

Because we used probabilities as feature weights, we can define a measure that combines such values into the probability $p(C|d)$.

We have defined a number of alternatives to estimate this probability, which we describe in the following.

Comb-inf-pos This measure uses the feature weight that we called inf-pos, as defined by (5.5).

Given a document, we consider the set D_F of features that are part of the category profile and that are found in the document. For each pair of subsuming features in D_F, we eliminate the most general one. Also, considering all the remaining pairs of features that share terms, we eliminate the ones whose weight is lower. With this step, we expect to make a better approximation about the independence of features upon which the document is classified.

Using the weights of each feature, as measured by inf-pos, we estimate $p(C|F_1, \ldots, F_n)$, where $F_i \in D_F$. Because we assume that each feature is independent, we make the estimation by the formula:

$$p(C|F_1, F_2, \ldots, F_n) = 1 - \prod_{i=1}^{n}(1 - p(C|F_i)) \tag{6}$$

where, as said, $p(C|F_i)$ is simply the inf-pos weight of the feature F_i.

With this formula, we interpret each feature as an independent event which can have success (the document belongs to the category) or failure (the document does not belong to the category). The document belongs to the category if at least one of the events succeeds, whose probability is equal to the inverse of the probability that all of them fail. Because events are assumed independent, we can evaluate the latter probability by the product of the inverse of the probability that they succeed.

Inf-pos-neg The approach we used for the definition of "inf-pos-neg" is different. The belonging of a document to a category does not only depends on the document and the profile of the category, but also on the profiles of the other categories.

For this purpose, the weight called inf-neg, as described in 5.6, are used, in order to diminish the probability that the document belongs to a category given that it shows big evidence of other categories for unusual features.

The "likelihood" that a document belongs to a category is estimated as above (the number, however, is not a probability):

$$Inf\!-\!pos\!-\!neg(C, e) = \frac{\sum_{i=1}^{p} p(C|F_{Ci})}{\beta + \sum_{i=1}^{np} p(\neg C|F_{\neg Ci})}$$

where:

- p is the number of features in the document that belong to the profile;
- F_{Ci} is the i-th feature found in the document and that belongs to the profile of C;
- $p(C|F_{Ci})$ is the value of the inf-pos weight associated to F_{Ci};
- $\beta > 0$ guarantees that the denominator is non-zero;
- np is the number of features found in the example that does not belong to the profile of C but to the one of other categories;
- $F_{\neg Ci}$ is the i-th feature found in the example that does not belong to C;

- $p(\neg C|F_{\neg Ci})$ is the value of inf-neg for the feature $F_{\neg Ci}$ relatively to the category C.

While this measure takes into account more features of the document (and we will experimentally verify how this is the most effective measure), in a way it "cheats" about profile compactness, since each feature has not just one weight but multiple, precisely, one for each category.

Inf-pos-single The measure that we called "inf-pos-single" is, so to say, "anomalous", and it shows how, depending on the domain, ad hoc measures can perform as well (or better) than standard techniques.

This measure privileges "short" features as opposed to "longer" ones, and it therefore makes sense only for structured features. In the evaluation of the likelihood that a document belongs to a category, the weight of features, as measured by inf-pos, is divided by the length of the feature, where by length we mean the number of terms in the case of feature of depth 1, and the number of noun phrases in the case of features of depth 2. The final estimation is evaluated, for all the rest, as by comb-inf-pos.

The reason why this measure performs well (it has better performance than comb-inf-pos, but lower than inf-pos-neg) is that shorter features are usually formed by a higher number of examples, and because a generalization which was not found ambiguous is likely to be a good one, it is "encouraged". Another thing to consider, in the case of feature of depth 2, is that while terms in the same noun phrase must be close together, different noun phrases could be far apart, and thus it is more "dangerous" to consider such a generalization scheme.

Take-all This function was used as an "experiment" to verify the system behavior at an early stage of development, but it nevertheless gave meaningful indications.

This measure simply corresponds to the inf-pos measure of the feature with higher weight found in the document. It is an underestimation of the probability that a document belongs to a category: it helped fine-tuning the system (e. g. defining inf-pos-single) when we realized that a document had a high chance to have at least a feature with high weight. This affected most of the probabilistic measures.

7.3.2 Thresholding technique

To determine thresholds it is necessary to scan a set of examples, thus performing a sort of training.

Starting from a set of documents, for each category the value of the selected matching function is evaluated for each document. Values are then ordered increasingly: at this point, different techniques can be used to determine the "cut-off" value the documents above which are assumed to belong to the category.

- A technique is called *proportional thresholding*, and it establishes a threshold so as to assign to each category the same proportion of the document on the threshold set that it has on the training set. Once again, the distribution of the documents in the training set is assumed to be a good sample of the documents which will have to be classified by the system. To vary the thresholds in a parametric way so as to affect the precision and recall aspects one can act on a parameter which multiplies the percentage of documents which will be assigned to the category, forcing the system to assign it more or less document as specified by a parameter higher or lower than 1 respectively.

- Another technique is the one called *probabilistic thresholding*. The cut-off is chosen so as to maximize performance over the threshold set. This is a wrapper-like method, but while one could think this would be the best way to go, it was shown experimentally that it tends to "over-fit" the threshold data, by maximizing the performance over it, but making the system less plastic.

It is not necessary to define thresholds in order to perform classification, and there are examples in literature where this opportunity was explored (see e. g. [15]). Usually this involves, however, some form of knowledge of how many documents to assign to a category, or to how many category assign the document to, and the like, which could not be generally available.

While a module to assign thresholds to each category, given profiles, matching function and threshold set has been implemented, for test purposes we did not use such thresholding techniques, since they introduced another parameter to take into account.

Tests were done by classifying documents of which we knew the percentage that had to be assigned to each category. We then evaluated the percentage of correctly assigned documents in that percentage. This gives an idea of how well the matching functions puts relevant documents on top, and it is a measure that can be compared to other system's retrieval evaluation.

We must consider, however, that the *stability* of a matching function is an issue as well. If the thresholds must be placed at a value for which the function takes very close values, the system is more likely to make mistakes than if the threshold falls on a value around which the function has sparse values.

7.4 Symbolic approach to classification

Another approach to classification is the use of symbolic techniques, more oriented to the logic than to the numbers.

With statistical approaches the weight of features is what matters for the decisional process, which combines all the weights of the features. This approach has the advantage that it is robust to noise in the data, since the combination process absorbed anomalities.

Symbolic methods, in the other hand, allow a better use of the semantics of the profiles. It is interesting to see how such profiles could be used to build a classification scheme based on symbolic manipulation instead of on statistical analysis.

7.4.1 Decision trees

A typical approach of this kind is *decision tree learning*. In this case, the profile of the category simply represents the set of features that are used to build the decision tree.

To build a decision tree it is necessary to define the *event* to classify, the attributes on which to base the classification rule and the evaluation function used to select the best attribute at each step. In literature there are many variants of algorithms to build decision trees, but each one of them requires the definition of this fundamental components. While in general many choices are possible, in this context we define:

- an *event* is a document, and it is an example for the learning process when the result of classification is known;
- the *attributes* are the features obtained by the learning process that built profiles.[12] Each

[12] The term "profile" is not proper in this context, since if the profile is defined as the description of the category meaning, then in this case it would be the decision tree itself. What we call "profile" is simply the set of features obtained by the training process.

feature will be treated as a binary attribute, which has value `True` if the feature is in the examined document, `False` otherwise;

- for the *evaluation function*, many choices can be made. The characteristics of the domain, however, suggest that the most widely used function in literature, the *information gain* [23; 20], might perform appropriately. This measure is the most natural one to use, considering that it takes into account the discriminant capacity of a feature relatively to the set and to the subset in which it would split it. As we already discussed for evaluation functions, different measures could be deployed. The decision tree learning technique is in fact very general, and can be "customized" in order to apply it to specific domains.

The most intuitive way to approach the problem is to build a decision tree for each category. The tree should determine whether or not a document belongs to the category independently from other categories.

In the end of the learning process for each category we will then have a *forest* of trees which represents their meaning. To apply the decision tree technique to our case, however, we can make some specific observation.

First of all, we must decide which attribute to use to build the description of each category. If, intuitively, we would use the feature set obtained from the training algorithm for the category, another solution is possible. While the profile represents the meaning of the category, it would not surprise if we needed something different for a better performance in classification. It is possible that, for characteristics of the domain, the "negative" information provided by some feature belonging to the profile of some other category is a better discriminant. The best choice to do will therefore be the union of all the features in all profiles.

As for the training process, it is possible that situation in which there are no more attributes available to split the set further occur. This is because it is not guaranteed that the features extracted allow a total distinction amongst documents of different categories. Should this situation occur, we will have to assign to the node a terminal value, depending e. g. on the classification of the majority of the documents.

A peculiarity that can be exploited is, in the case of feature of depth greater than 1, the subsumption relationship. Since not all attributes are independent, once an attribute has been chosen, it is possible to eliminate all the ones whose value is implied for the construction of the relative subtree. This optimization does not affect the quality of the constructed tree.

7.5 Evaluating classification

Many measures have been introduced to evaluate the quality of a classifier, but they all rely on the same model. Such model is based on the definition of a *contingency table* [15]. In the contingency table we represent the decisions taken by the classifier, splitting apart the correct ones from the wrong ones. The classifier is modeled as a system to which n binary questions are asked. Questions come in the form: "does document d belong to the category C?" For each decision the classifier takes, there are four possible cases:

- the correct decision is "yes" and the classifier decided "yes" (the number of such decisions being a);
- the correct answer is "no", but the classifier decided "yes" (b is the number of these cases);
- the correct answer is "yes", but the classifier said "no" (c is the number of these cases);

- the correct answer is "no" and the classifier said "no" (d is the number of these cases).

We will obviously have that:

$$a + b + c + d = n$$

if n is the number of documents to classify. Two measures are defined over the contingency table: *precision* and *recall.*

Precision is defined as:

$$precision = \frac{a}{a+c},$$

that is, an indication of how much "garbage" was assigned to the category (the lower precision, the more garbage), while recall is defined as:

$$recall = \frac{a}{a+b},$$

an indication of how much "good stuff" was mistakenly left out from the category (the lower recall, the more is left out).

From the definition it is obvious that recall and precision are two different aspects, both of which have to be taken into account. A classifier can achieve high recall by assigning all documents to the category, but it would, in this case, achieve a precision very close to 0. Conversely, by assigning very few document to the category[13] (only the ones for which the decision is easier), a classifier would easily achieve a precision close to 1 in spite of recall, which would drop down to a value close to 0.

To combine both measures, a function called $\mathcal{F}_\beta$ is defined. Its definition is as follows:

$$\mathcal{F}_\beta(p,r) = \frac{(\beta^2+1)\cdot p \cdot r}{\beta^2 \cdot p + r},$$

where p and r are precision and recall respectively. A definition in terms of a, b, c and d is also possible:

$$\mathcal{F}_\beta(a,b,c) = \frac{(\beta^2+1)\cdot a}{(\beta^2+1)\cdot a + b + \beta^2 \cdot c}.$$

The value d is not considered, since the "silence" is, in normal condition, too much of a frequent event to be considered as a positive decision.

The value of β is usually set to 1, so that the function gives precision and recall the same importance. In this case, its definition is simplified to:

$$\mathcal{F}_1(a,b,c) = \frac{2\cdot a}{2\cdot a + b + c}.$$

There are two ways to use this function. Because it only evaluates decisions relative to the assignment of documents to one category, it is necessary to put these values together to evaluate the performance over the whole set of categories. One way to do so is called *microavereged.* With this approach, we put all the decisions relative to different categories together in the same table, and we evaluate $\mathcal{F}$ over that table.

[13] "Very few" has to be adjusted to the number of documents that originally make it into the category.

Another approach is called *macroaveraged*, with which we evaluate $\mathcal{F}$ separately for each category, then averaging over all $\mathcal{F}$'s to obtain the final measure.

The first technique gives equal importance to each document, so that the more documents are classified correctly, the better, while the second approach gives equal importance to each category, so that all categories are considered the same, no matter the number of documents they contain.

Although the easiest way, it is not always desirable to have all binary decisions. Many times, and especially for the evaluation, it would be interesting to consider a finer approach, so as to know, in case of error, by "how much" we missed the correct answer. Such an approach would make everything more complicated, however, and it has therefore not been studied here.

The evaluation techniques discussed require the knowledge of the correct classification for each document in the test set. Because such a classification is usually performed by hand, we must take into account that human classification is far from perfect. Many times the answer of the classifier could differ from the one given in the test set, which is considered as an error, when the human decision is opinable or just as justifiable as the system one.

8 Implementation

We implemented the proposed model, in the described instantiations, so as to make tests and evaluate experimentally the validity of our research. We believe that research, for theoretical it can be, should always aim problems directly. They do not need to be ground or real problems, so as to cause the research to be too much domain-dependent, but they need to be what directs research, instead of an optional test field on which to force a study that has kept on an abstract and self-feeding level.

When tests and experimentations are available, the effort to try a model in reality should always be made, even by just building a simple and non-distributional prototype (like we did). We have found that except for the very basic idea, experimentation was what lead the research mostly, and gave the most valuable hints about what to do.

When experimentation is not possible, theory better be well founded when not proven. Exploiters of our studies should always know without an effort on their side whether what we have done works or not, so that they can decide whether or not to follow our approach. When a research gives poor results, it is still a valuable research, since it makes researchers concentrate on the right things immediately.

The prototype we developed was used throughout the work, and was designed to allow us to change the many parameters we described in order to fine tune the system. For this fast prototyping necessity, we decided to implement the system in *Lisp* [8], since it minimized coding time, and its interactiveness gave us a valuable tool to do experimentation. Amongst the many common Lisp implementations we chose *ECoLisp*: *Embeddable Common Lisp*. ECoLisp is written and maintained by Giuseppe Attardi, and it is a full Common Lisp interpreter and compiler, with CLOS facilities. It gives the user a chance to write C functions and embed them into the Lisp program, which was a great way of extending the program with extra-Lisp features, such as a graphical interface, and the like.

To monitor the program activity, as well as to make development faster, we used a version of ECoLisp that we "hacked" so as to include graphic support as provided by *Tcl/Tk* [21], and that we called *ECL/TK*. This gave us a fast and complete way to build our own interfaces quickly.

A porting to Java and/or C++ is planned. We will try and make the system into a reusable component, to be inserted and deployed inside bigger tools. We believe that with a reasonable interface, such a component could be of help to explore other techniques, or extend the present model. Performance improvement (in time) is planned too, when porting the program.

There is a part of the system currently not written in Lisp. The interface with the tagger, and the noun phrase extraction is done in Java, since a skeleton client was already available.

8.1 System architecture

The whole system is composed by a set of separate tools which are intended to work in pipeline. Every module produces the data necessary to the following one. Some of the modules are optional and can be skipped.

Communication amongst the various modules is done trivially by taking the previous module's output as part of the input, in the form of a file. An exception to this rule are the graphical monitoring modules, which were implemented as separate processes not to burden the training module down, and which take input via an asynchronous stream (a socket) with the training module. The stream can be as asynchronous as being a file, so that logs can be created with no cost at development time. The use of a socket allows us to eventually move the graphical process on another machine, in order to leave the server on which the learning is running lighter.

The system is organized as shown on figure 10.

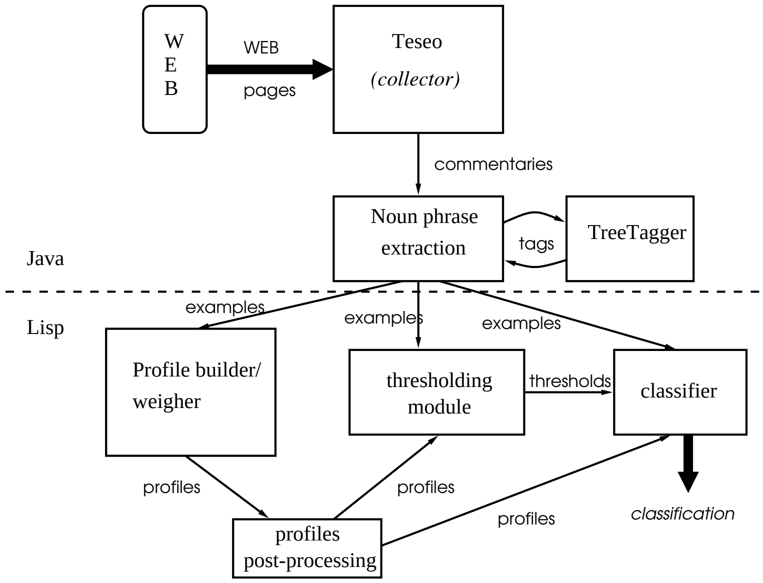

Figure 10: **the architecture of the system. Monitor components are not shown: their interactions only happen with the module they are monitoring.**

8.2 Acquiring documents and querying the tagger

The first module is the one that performs document acquisition from the Web, collecting the commentaries we described earlier. Such a module was not written from scratch; instead it was implemented as a Teseo [3] component. Because Teseo was implemented so as to allow this kind of modifications, it saved us probably about a month of work, showing the value of making components available to the community.

This module performs the following tasks:

- exploration of the Web and retrieval of documents;
- for each commentary extracted, assignment of the correct category as specified by the source (this is done by Teseo's filters arranged for the most popular search engines);
- tagger interrogation, and transformation of the commentary in list of noun phrases.

The assignment of the category to each commentary is done exploiting the informations provided by the selected source. The technique we used (Teseo's) depends on the source; filters have been written in Perl, so that, for example, from the title of a page of Virgilio it is possible to determine the category to which the contained commentaries belong.

This is the module to modify when we want to give a different definition of a noun phrase, or if we decide not to use the tagger.

The output of this module is a text file containing the commentaries extracted as well as their classification. For this purpose, we used the following syntax:

```
CAT <category-list>
CON <commentary>.
```

We used the tag `CAT` to indicate the category to which the commentary belongs. The tag `CON` "introduces" the list of noun phrases in which the original commentary has been transformed.

Every term is represented by a pair (actually a "cons") whose elements are the lemma and the tag as indicated by the tagger.

We give an example of how a commentary is transformed. The page containing the link is part of the catalog of Virgilio, and has the title: *"Siti italiani - Arte e Cultura - Musei e gallerie - Scienze e tecnologia"*.[14] From the title, we took the information that the main category to which the document belong is "Arte e Cultura" (art and culture).

The commentary appeared as follows:

Museo Civico di Storia Naturale di Morbegno

Tutte le informazioni: ubicazione, orari di apertura, collezioni, pubblicazioni scientifiche, ricerche necessarie per conoscere il Museo Civico di Storia Naturale di Morbegno.[15]

This is the commentary as found on the Web page. The transformation turns it into:

```
CAT (ARTE_E_CULTURA)
CON (((museo . NOM) (civico . ADJ) (storia . NOM) (naturale . ADJ)
      (morbegno . NOM) (tutto . ADJ))
     ((informazione . NOM))
     ((ubicazione . NOM))
     ((orario . NOM) (apertura . NOM))
     ((collezione . NOM))
     ((pubblicazione . NOM) (scientifico . ADJ))
     ((ricerca . NOM) (necessario . ADJ))
     ((museo . NOM) (civico . ADJ) (storia . NOM) (naturale . ADJ)
      (morbegno . NOM))
     ((museo . NOM))
     ((morbegno . NOM))).
```

[14] Examples will be in Italian since they are taken from real pages.

[15] In English: "Civic Museum of Natural History of Morbegno - All the informations: location, business schedule, collections, scientific publications, researches necessary to know the Civic Museum of Natural History of Morbegno".

We divided the commentary over different lines to make the reading easier, while that is not done by the actual module.

8.3 Identifying duplicates and splitting

There is a phase of post-processing over the extracted documents that has two main functions: first, duplicates are looked for and eliminated, second, examples are distributed over different files for use as training, threshold or test set.

During the collection of the commentaries, it is possible that the same commentary was collected multiple times. If we think of the organization of a catalog-organized search engine, we see how, spidering the site, we would find the same commentary on different pages, as it may belong to different categories. Because our system, when told that a document belongs to a category, implicitly assumes that it does not belong to other categories (as imposed by the ambiguity resolution), it is necessary, when inserting a document, to tell the system about all the categories to which it belongs. It is therefore important to scan the input looking for duplicates, reuniting them in the same case. Thanks to the use of hash tables, this if done in $O(n)$, where n is the number of documents. It is important that the system does *not* require in general the availability of the whole training set beforehand: it is the idiosyncrasy of the source that forces us to introduce a preprocessing step.

Once duplicates have been eliminated, documents are simply divided amongst three files according to a percentage that can be specified by the user.

8.4 Learning system

This is the main module of the system, and it implements the algorithm as described in section 3. It takes as input one of the files of output of the previous module, and returns as output the profiles of the categories for which documents have been examined.

The *CLOS* allowed us to build a common skeleton for the main part of the algorithm, while all that depends on the feature structure was implemented as an object or a method, so that different instantiations could be done by simply rewriting the parts of the algorithm that changed. We can say that the model is implemented as a set of functions, while instantiation parameters are implemented either explicitly (as numerical parameters) or with the use of objects.

The categories are organized as objects, as follows:

```
Class: category
Parents: none
Attributes:
   name        ;name of the category
   profile     ;list of features: profile of the category
   examples    ;list of examples for the category
   n-ex        ;number of examples
   n-doc       ;number of documents
   attributes  ;list of attributes of the category
```

The implementation of features is done via a hierarchy of objects, which was exploited for the only reason to make changes in feature structure easier. Flexibility has been obtained by upcasting.

```
Class: feature
Parents: none
Attributes:
   description   ;description of a feature/example
```

This class was used to create three subclasses for the three different instantiations we gave at depth 0, 1 and 2. The subclasses are formally identical, and their structure is as follows (for example, we take the depth 0):

```
Class: feature-0
Parents: feature
```

The information about the kind of feature is stored in the type itself, so that upcasting will fire the appropriate method. As evident from the definition of the model, the algorithm is made parametric in the feature structure, which therefore does not have to appear explicitly.

To represent the features that are part of a profile (and then have a threshold and other data) we defined a class `profile-feature`, which is defined as follows:

```
Class: profile-feature
Parents: feature
Attributes:
   examples   ;list of examples subsumed by the feature
   n-ex       ;number of examples in the above list
   threshold  ;specificity threshold
   category   ;category of the feature
   modified   ;T iff the feature was modified
   index      ;a progressive number different for all features
   attributes ;associative list holding auxiliary data
```

The attributes `examples` and `n-ex` are used to do reverse searches and make the process faster.

The instantiation of the three subclasses is done as in figure 11.

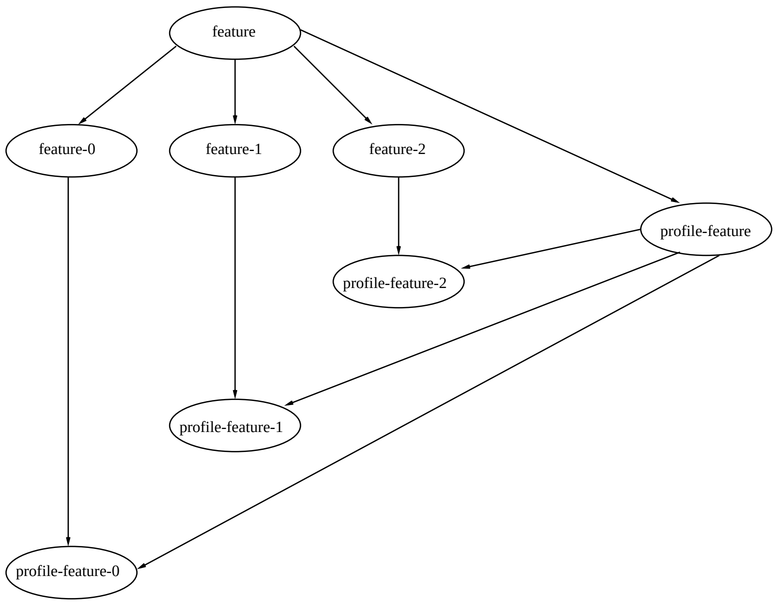

Figure 11: **class hierarchy.**

With this structure, the class of profile features of depth 1 has the following simple structure:

```
Class: profile-feature-1
Parents: feature-1 profile-feature
```

The same hierarchy is defined over examples, where instead of the class `profile-feature` there is a class `example` with the following structure:

```
Class: example
Parents: feature
Attributes:
   index   ;progressive number different for all examples
   cats    ;list of categories to which the example belong
```

With this class as parent, three subclasses are defined: `example-0`, `example-1` and `example-2`, as already done for the profile features.

The use of multiple inheritance is superfluous, and could be avoided, since the same exact effect could be obtained by upcasting. We decided to leave the scheme as it is, in the prototype, to be able to explore different solutions in the future, but it would be stripped out in an eventual porting (also, Java does not allow multiple inheritance).

All the functions which depend on the structure of the features were implemented as methods, so that different definitions of the same method with different parameters makes it possible to overload the operator with different functions according to the type of features. This allowed a completely independent definition of the algorithm.

For example, we defined a method `generalize` which takes a `profile-feature` and an `example` to perform generalization. A redefinition of this method for `profile-feature-2` and `example-2` will change the system behavior.

During the training, basic information about the process on the run is output to standard output. For each document, the following informations are provided:

- progressive number of the document;
- the category to which the document is associated;
- whether or not the examples in the document were "captured": for each "new" example, a "*" is printed;
- the elements of the example that were recognized into the category profile;
- the percentage of the length of the example that was recognized.

An example of the information output is the following, obtained with features of depth 2 (for which documents equal examples):

```
2349 AMBIENTE_E_NATURA * NIL (0%)
Was: ((MAGNIFICAT__NOM MOSTRA__NOM INTERNAZIONALE__ADJ FELINO__ADJ
       CITTA__NOM ROMA__NPR))

2350 ARTE_E_CULTURA  (((GALLERIA__NOM)) ((PITTORE__NOM))) (14%)
2351 MUSICA * (((DISCOGRAFIA__NOM)) ((ALBUM__NOM))) (9%)
```

The example number 2349, belonging to the category `AMBIENTE_E_NATURA` was not "captured" by the profile of the category, and a new feature has then been created (as indicated by the symbol "*"). There were no recognized features (as indicated by `NIL`), and the percentage of the document that was recognized was then 0%. In this case, the unrecognized example is printed, so that we can look at it, and see if it reasonable that the example was hard to recognize.

The situation is different for the document 2350: this document, belonging to the category `ARTE_E_CULTURA`, had two features recognized: `((GALLERIA__NOM))` and `((PITTORE__NOM))`. No new feature was created, since 14% of the document was "captured" (10% is the limit for a document to be considered new).

The document 2351 shows a third situation. Two features were identified inside of it, but the percentage of the "captured" document is below 10%. For this reason, a new feature is created out of the document.

Once the learning is over, it is possible to apply the consolidation techniques we described in 6.1.1 and in 6.1.2: in particular, in this phase we post-generalize and eliminate features that have fewer than a fixed number of examples.

It is then possible to proceed with a weighting technique by using some of the functions described elsewhere. More than one measure can be used at the same time.

After the weighting is done, it is possible to apply the second kind of reduction techniques: the one based on the weight.

Profiles are finally saved to disk to make them available to other modules: the task of learning descriptions is, at this point, completed, and profiles can be read.

8.4.1 Monitoring the learning process

Further information is available if we want to have more details about the training on run. To allow a more complete analysis, which was very useful during the development phase, we implemented a module specifically to display various data about the training on the run.

As for global informations, two kinds of data are shown:

- the total number of features in the profiles;
- the probability that an example will cause the creation of a new feature (which is the probability that a new example is not "captured", and is interpretable as the learning momentum).

These two data are shown numerically and by a graph which represents their change in time (as more examples are observed).

The probability that an example causes a new feature is estimated considering each example as an aleatory variable X_k. The examples are therefore a succession of binary events X_i ($\{X_i\}_{i\in\mathbb{N}}$). The probability that the k-th example will cause the formation of a new feature is represented by $p(X_k = 1)$. We can then approximate the probability with the succession:

$$p_0 = p(X_0 = 1) = 1$$
$$p_k = p(X_k = 1) = p_{k-1}\alpha + (1-\alpha)X_k,$$

where α represents the stability we impose to the succession. Usually, values of α close to 1 are used, such as 0.995.

For each category, it is possible to obtain statistics about the profile. The informations shown are:

- percentage of the examples belonging to the category relatively to the total number of examples examined so far;
- percentage of the features that are in the profile of the category relatively to the total number of features in all profiles;
- histogram with the distribution of the features relatively to the time at which they were created. We define the concept of "age" of a feature by identifying its "birth-date" as

the progressive number of the example it was created from. A feature is old when it was created many examples ago, and has therefore spent a long time in the profile without having been judged ambiguous. For each age interval represented as a column in the histogram, it is indicated the percentage of features which are still in the profile and the percentage of features which "died".

The structure of the interface is shown on figure 12. On the left side we see general informations about the set of all features, while the right part is reserved for category-specific data. One changes the category by pushing the buttons indicated by "<<" and ">>" in the figure. The scalers beside the graphs allow to change the scale of the graph in the window.

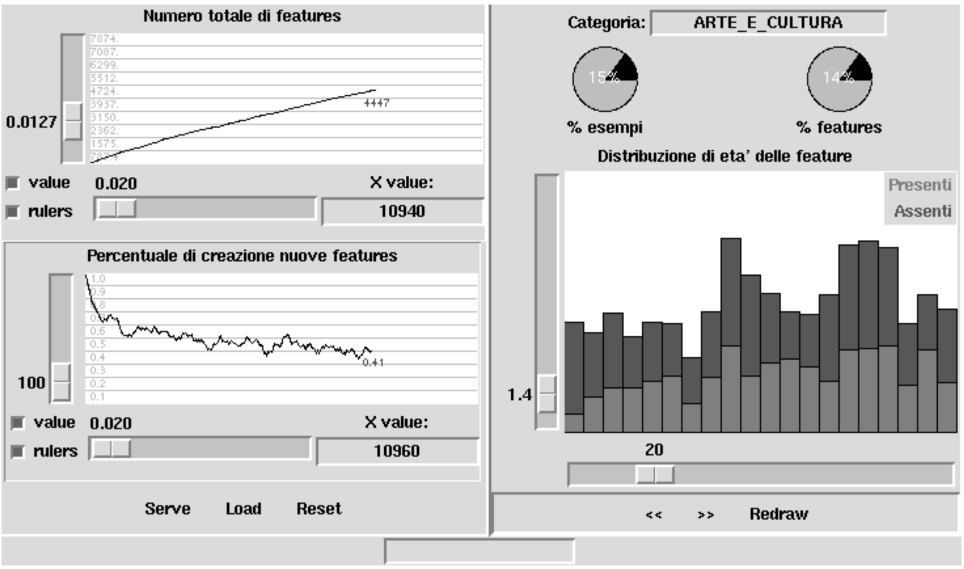

Figure 12: **interface of the monitoring module.**

8.5 Document ordering

This module evaluates the capacity of the profiles to impose an ordering of the documents that puts the relevant documents on top.

The process is implemented as follows:

- a set of example is given to the classifier;
- for each document and for each category, the retrieval value is calculated by the specified matching function;
- for each category, documents are ordered decreasingly;
- the ordering obtained is evaluated.

To evaluate the ordering, we consider the number of documents that should be assigned to the category out of the given set off the top of the list and we evaluate how many of such documents actually belong to the category. To have a visual feedback of the classification quality, we have developed a graphical representation, which is visible in figure 13.

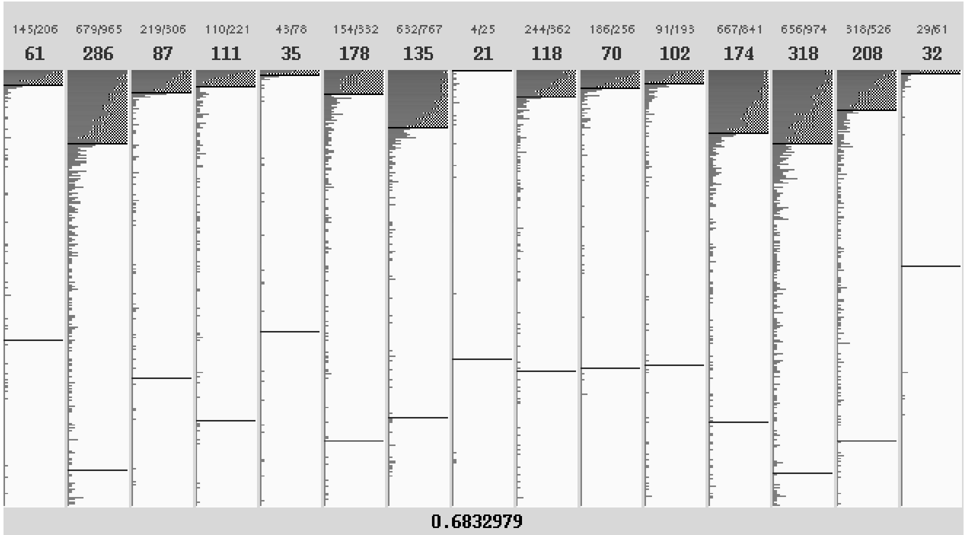

Figure 13: **interface for the module for document ordering.**

The various categories are represented on different columns. Each column is visualized when the computation for the category is over. The informations shown for each category are:

- ratio between the relevant retrieved documents and retrieved documents;
- number of documents belonging to the category which were "missed", which is also the number of irrelevant documents retrieved;
- graph of the sorted documents: the band width represents the density of relevant documents, while the shaded area represents the optimal sorting (the band should have the shape of the shade for 100% accuracy).

The more overlap there is between the band and the shaded area, the better the accuracy is: the missed shaded part is nothing but the classification error.

In the lower part of the window a global evaluation number, $\mathcal{F}_1$ microaveraged, was used to combine the values.

Because this module computes the retrieval value for each category and document, it can also be used to assign thresholds to each category. By selecting a threshold strategy, it will save a state made of the profiles it took in input, plus a threshold for each category. If multiple matching functions are specified, only the first is graphically represented, but thresholds are evaluated for all functions. Each category will therefore have a threshold for each matching function.

8.6 Classifying with thresholds

This module realized the classification with the use of thresholds.

It takes as input a matching function, category profiles and thresholds as output by the previous module, and a set of documents to classify. The classification results are written to a file for later inspection, as well as a log-file which keeps track of the *reasons* why a document was classified in a certain way.

The classifier accepts some parameters upon which to determine a classification strategy which can go beyond the simple threshold-oriented technique. Examples of such techniques are:

- assign to document to the same number of categories the document is said to belong to: this information is usually not available, but in cases where the number is fixed, it could be a strategy. Such a method has been used for example by Lewis in [15], and it requires an absolute measure, in order to compare categories;
- assign the document to at most n categories: the best n categories for which the threshold is surpassed are supposed to own the document. This technique was used again by Lewis [17; 15];
- *recover*: in case no category surpasses the threshold, the document is assigned to the category whose value is closest to its threshold. This is a useful technique when we know that the document is assigned to at least one category.

In the log-file generated by the classifier, informations about the correct category, assigned category with the retrieval value and the threshold, the features found in the document with their weight are shown. When no category has been chosen, the two categories closer to their thresholds are shown.

When the correct category has not been chosen, the value obtained for it is reported together with the threshold.

In the following we can see some examples. The matching function used for the examples is "comb-inf-pos".

```
Documento: (((ANDREA . NPR) (CAMPANINI . NOM))
            ((SITO . ADJ) (PERSONALE . NOM))
            ((DEDICARE . VER-PPER) (WINGTSUN . NOM))
            ((ARTE . NOM) (MARZIALE . ADJ))
            ((PRATICO . NOM))
            ((CAMPANINIP . NOM)))
Categoria/e corretta/e: (SPORT)
SPORT  0.9 (0.862)
((ARTE__NOM MARZIALE__ADJ)): 0.9
```

The correct category `SPORT` has been assigned, as the matching function has value 0.9 while the threshold is 0.862. The recognized feature is `((ARTE__NOM MARZIALE__ADJ))`, whose weight is 0.9.

```
Documento: (((CAMPIONATO . NOM) (ITALIANO . ADJ) (CALCIO . NOM)
             (PROGRAMMA . NOM) (PROFESSIONALE . ADJ)
             (GESTIONE . NOM) (CAMPIONARE . VER-PPER)
             (CALCIO . NOM) (SERIE . NOM))
            ((UTILIZZARE . VER-PPER) (GIORNALISTA . NOM)
             (SPORTIVO . ADJ))
            ((SOCIETA . NOM) (SERIE . NOM))
            ((STATISTICO . ADJ) (DETTAGLIATO . ADJ)
             (SQUADRA . NOM))
            ((GIOCATORE . NOM))
            ((CALCOLO . NOM) (AUTOMATICO . ADJ) (CLASSIFICA . NOM)
             (AVULSO . ADJ))
            ((SUPPORTO . NOM) (PRONOSTICO . NOM)
```

```
             (TOTOCALCIO . NOM))
            ((TOTOGOL . NOM))
            ((AGGIORNAMENTO . NOM) (AUTOMATICO . ADJ)
             (TABELLINI . ADJ) (VIA . NOM) (WEB . NOM))
            ((E-MAIL . NOM)))

Categoria/e corretta/e: (COMPUTER_E_INTERNET)
SPORT   1.0 ( 0.862)
((SOCIETA__NOM)): 0.4
((CAMPIONATO__NOM) (SQUADRA__NOM)): 1.0
((GIOCATORE__NOM)): 0.7
((CLASSIFICA__NOM) (CALCIO__NOM)): 1.0
((TOTOCALCIO__NOM)): 0.4
((PRONOSTICO__NOM) (TOTOGOL__NOM)): 0.5
INFORMAZIONE  0.713 (  0.59)
((GIORNALISTA__NOM)): 0.6
((AGGIORNAMENTO__NOM)): 0.1
Corretta non scelta COMPUTER_E_INTERNET  0.517 ( 0.631)
```

In this case we see an example in which the document was not recognized. the correct category is `COMPUTER_E_INTERNET`, while the two better categories found are `SPORT` and `INFORMAZIONE`.

This tool allows us to control not just the final results, but case by case the classification events. It was then easier to understand what went wrong, and to develop a way to fix the problem.

To evaluate the performance of this classifier, we have developed another small module which takes as input the classified documents and the test set. By comparing the two classifications, the module evaluates the $\mathcal{F}_1$ measure as defined in 7.5.

9 Results

The analysis of the results is done in a series of phases which analyze the various aspects about profile construction and their use. It is not possible to evaluate objectively the quality of profiles, but when possible, like for classification results, this has been done.

The first analysis has been done on the training process, to examine the evolution of the state. After the training process, when profiles are definitely constructed, we subjectively examined the profiles to evaluate the quality of the information extracted. The final phase of the analysis was done by using standard evaluation techniques on classification results.

9.1 Analysis of the collection deployed

The documents we used as training and test sets where obtained from Virgilio's catalog [31]. Such a catalog is organized hierarchically as a category tree of depth (for the analyzed part) 3. We choose to only take the first level into account, so that all the deeper nodes are moved up to the first level. This decreased the number of categories, making an evaluation easier, and at the same time made their meaning more general and thus "hard" to represent, while the number of examples for each category was drastically higher.

The categories taken into account are 15, and are indicated in figure 9.1. The total number of documents considered is about 20000. Of these, 11000 were used for the training, while the remaining were split into various test sets.

Category name	Percentage
AMBIENTE_E_NATURA	3.49%
ARTE_E_CULTURA	14.77%
CINEMA_E_SPETTACOLO	4.70%
COMPUTER_E_INTERNET	4.00%
INFANZIA_E_RAGAZZI	1.19%
INFORMAZIONE	5.74%
MUSICA	12.41%
POLITICA_E_SOCIETA	0.76%
SALUTE_E_MEDICINA	5.96%
SCUOLA_E_ISTRUZIONE	4.32%
SHOPPING	3.45%
SPORT	13.23%
TEMPO_LIBERO_E_CASA	16.21%
UNIVERSITA_E_SCIENZE	8.62%
VIAGGI_E_TURISMO	1.14%

Figure 14: **list of the categories considered. For each category the percentage of documents belonging to it is indicated. There is overlap between the categories.**

The distribution amongst the various categories is not uniform, and there is a big difference between the documents in the most populated categories, for example TEMPO_LIBERO_E_CASA, and the more rare categories, such as POLITICA_E_SOCIETA. This allows to verify the system behavior in these circumstances.

The correct classification is obtained from the title of the page, and it is therefore due to Virgilio's experts. These catalog-organized search engines make available a large human-generated source of information, which, if deployed automatically, could be a good knowledge source for the training of an intelligent agent. Because the classification is done manually, however, subjectivity is evident in some cases, which is the proof that no 100% accuracy can be achieved.

By using only one source, the profiles will be affected by the interpretation that the source gives about the meaning of categories. Further planned work includes the ability to integrate different sources, so that the meanings extracted are not affected by a single source's idiosyncrasies.[16]

9.2 Analysis during the training process

We studied the evolution of the profiles during the training process. We took two things into account:

- compactness of the profiles, and necessity to create new features in order to capture new examples;
- internal structure of the profiles, relatively to the characteristics of the features and to their evolution in time.

[16]In the case we considered, the source was already constituted of multiple experts, so that idiosyncrasies could be dampened. It is probable, however, that Virgilio's "bosses" gave precise instructions to the experts, the decisions of which thus still depend on the source.

9.2.1 Compactness of the profiles and training

We keep the dimension of the profiles under control, as compactness of the profiles is requested, to impose an effective generalization. In figure 15 we see two graphs obtained with a training

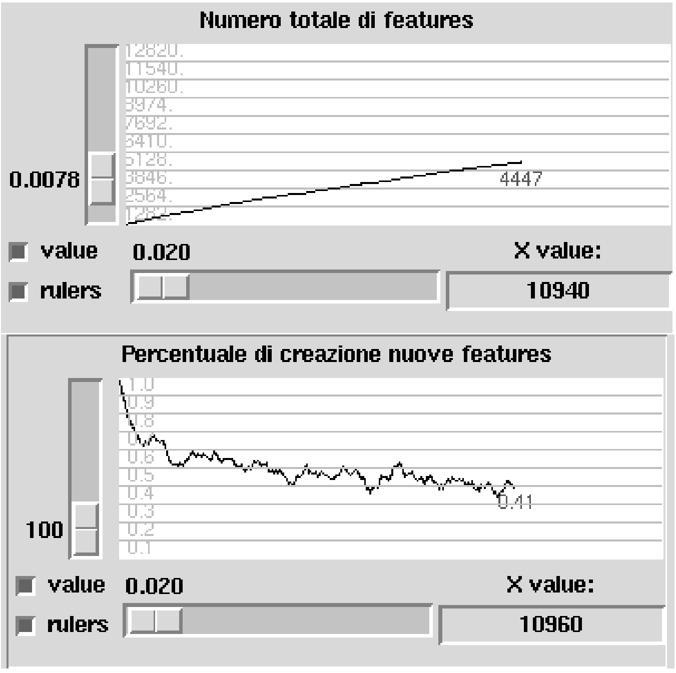

Figure 15: **total number of features and percentage of creation of new features.**

over 10967 documents with features of depth 2.

The number of features in the profiles of the categories is 4447. Amongst these, however, there are many features that will be eliminated by one of the cleaning methods described. Such features are ~ 600. Using the same training set, but different feature structure (noun phrases or terms), we obtained the following results:

- terms (depth 0): about 8700 features (counting only features that were found on at least 3 examples);
- noun phrases (depth 1): about 10000 features (counting only the features found on at least 3 examples);

In figure 16 see see how the total number of features in the profiles increases as more documents are examined. In our tests, the dimension of the profiles increased in a linear fashion. It is also evident, however, that the proportionality constant is greater for features which have depth 0 or 1 as opposed to the case of features of depth 2. This is also due to the fact that the same number of documents provide more examples at depth 1 than they do at depth 2.

In the case of depth 0 the number of features is equivalent to the number of terms for each category seen so far. We would expect the frequency of a new term to decrease as more documents are seen, so because the slope did not change, we will assume all the slopes for all depths could stabilize after more documents.

The case of features of depth 1 is surprisingly not in between the case of depth 0 or 1. Apparently, the system build more features at depth 1 than at depth 0 or 2. Although we do

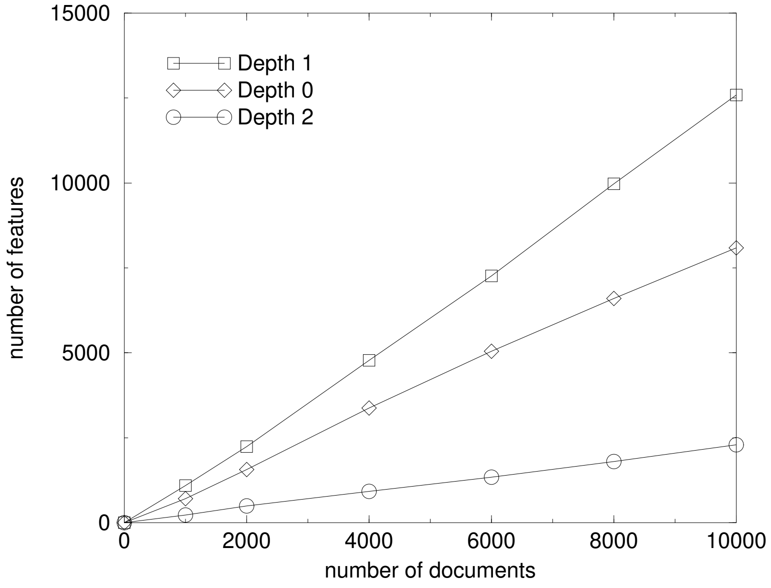

Figure 16: **number of features in the profiles as more examples are examined.**

not have a proof of our believes, we suspect that this is due to the greater generality capability that the system has at depth 2 and the allowed ambiguity at depth 0. Ambiguous terms are left untouched, while ambiguous noun phrases are split into two or more features, thus causing the number to increase.

The number of features we obtain at depth 2 is interesting. The number increases slower than at depth 0 or 1 (also because fewer examples are examined), thus giving us more concise descriptions. Should these descriptions prove to be good ones, we would have obtained one of the proposed goals.

The percentage of examples that are not "captured" gives us an important indication of the learning capacity. As shown on figure 15, where the feature of depth 2 case is shown, the percentage slowly drops down from 100%, the obvious value when the profiles are empty, at the time the first example is shown, to a value around 40% which is still considerably high. This means that the system is still in a phase of heavy learning, and could therefore benefit from a larger set of examples. On the other hand, we see that such a percentage drops down quickly from 100%, which means that large improvements are done after just a few documents. The roughly monotonous slope of the percentage shows how the system learns monotonically.

Should the mentioned percentage be too high, it would mean that the system does not generalize effectively, as new examples are never "captured", the system would keep collecting all examples. On the other hand, should the percentage be very low, that would mean that either:

1. the system learned everything perfectly, and can classify every document correctly. The learning phase is then impossible to take on, since no document can change a perfect state;

2. the system classifies every document in the category erroneously. Generalization has been excessive, and no document changes the state as it is "captured", be it relevant or not.

Both the cases of high and low percentage are therefore to be treated carefully: the best way is to try and keep a low descent of the probability, so as to make sure the system is actually learning correct descriptions. The examination of this percentage allowed us to define the predicate "*Captures*" effectively.

9.2.2 Internal structure of the profiles

We can also use our category-oriented tool to help us interpret the training process. We will proceed by showing an example.

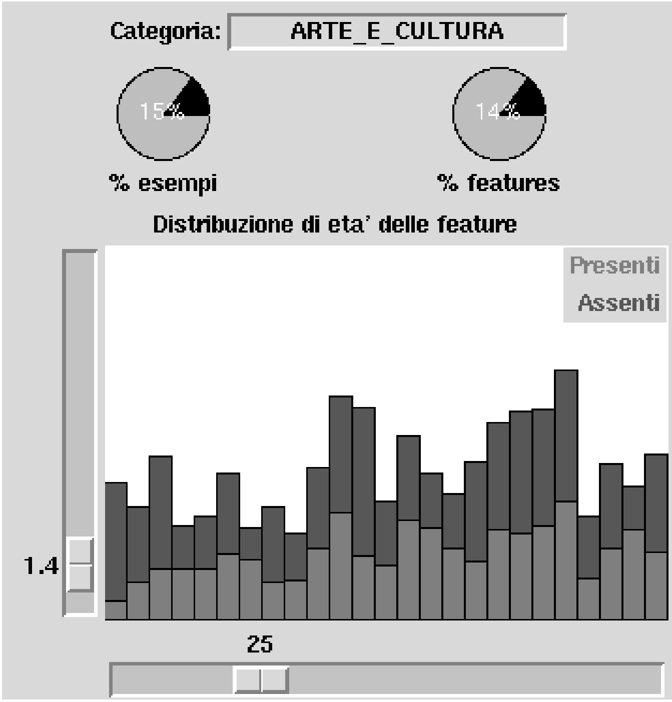

Figure 17: **data about the category** `ARTE_E_CULTURA`**.**

The figure 17 allows us to see the age of the features that make up the profile of the category `ARTE_E_CULTURA`. This category is one of the most populated, as it contains 15% of the documents. Features are divided into 25 bands of age. Each band is divided in two parts, the brighter one representing the feature that are still in the profile, and the other one representing the features that were specialized.

This spectrum shows us how while profiles change constantly, "old" features are kept if they prove to be valid ones. This is an important fact, because one could argue about the system being unstable. This graph shows that when a good feature is found, it is kept and it survives many examples.

9.3 Profile analysis

As we wanted to extract information explicitly, our profiles turn out to be *readable*. The simplest way to evaluate them is then to read the profiles and see if the match the category meaning.

Also, by reading the profiles we can have a better idea and verify of the task performed by the consolidation techniques we discussed in 6.1.1.

Let us consider the descriptions we obtained for the same category, but deploying different feature depths. In the following, we will show part of the profiles we obtained for the category

MUSICA[17] by examining the documents extracted from Virgilio. For each feature in the profiles we indicated the number of examples that are subsumed by the feature in the category and the weight as measured by inf-pos.

This part of the profile with features at depth 0:

```
(MUSICA__NOM 531 0.7334254)
(SITO__NOM 385 0.1599502)
(MUSICALE__ADJ 348 0.8592592)
(GRUPPO__NOM 291 0.399177)
(ITALIANO__ADJ 272 0.1635598)
(ROCK__NOM 203 0.9022222)
(BAND__NOM 200 0.9756098)
(SITO__ADJ 199 0.1776786)
(CONCERTO__NOM 188 0.8245614)
(UFFICIALE__ADJ 165 0.279661)
(DEDICARE__VER-PPER 161 0.1668394)
(PIU__ADV 146 0.173191)
(INFORMAZIONE__NOM 141 0.1036765)
(FOTO__NOM 137 0.2795918)
(PAGINA__NOM 136 0.1852861)
(JAZZ__NOM 130 0.9352518)
(CANZONE__NOM 129 0.8657718)
(ASSOCIAZIONE__NOM 106 0.1093911)
(DISCOGRAFIA__NOM 104 0.9719626)
(NOTIZIA__NOM 94 0.1657848)
(CORO__NOM 93 0.9587629)
(STORIA__NOM 90 0.1595745)
(GRANDE__ADJ 89 0.2730061)
(TESTO__NOM 87 0.4915254)
(MPT__NOM 84 0.9032258)
(BRANO__NOM 83 0.8556701)
(MONDO__NOM 77 0.1022576)
(TUTTO__ADJ 76 0.141527)
(THE__NOM 74 0.3794872)
(BIOGRAFIA__NOM 73 0.5658914)
(MUSICISTA__NOM 72 0.9)
(REPERTORIO__NOM 71 0.8554217)
(ANNO__NOM 70 0.2)
(LINK__NOM 70 0.1643192)
(BLUES__NOM 69 0.92)
.......
.......
```

The terms which appear in the profile are not unrelated to the category, but their meaning is often ambiguous. As an example, we can consider the term GRUPPO (group). While this can be put in relation to the category, its meaning is too general to be clear. For this reason, its weight is not too high.

Some of the terms are completely unrelated to the category, as SITO (site) and ITALIANO (italian) which appear just for their frequency.

[17] Examples will still be in Italian. We believe that their meaning is clear, nevertheless. We do not have English experimentation available for now, as the module that extracts noun phrases needs to be adjusted, but there should be no reason why what we will see should not be valid for English texts. As a porting will be completed, it will feature English support as well, and it will probably also have a language-independent modality.

At depth 1, the profile we obtain is as follows:

```
((BAND__NOM) 201 0.9804878)
((JAZZ__NOM) 131 0.9424461)
((CANZONE__NOM) 130 0.8724832)
((DISCOGRAFIA__NOM) 105 0.9813084)
((CORO__NOM) 94 0.9690722)
((MPT__NOM) 85 0.9139785)
((GRUPPO__NOM MUSICALE__ADJ) 74 0.9866667)
((MUSIC__NOM) 54 0.739726)
((CHITARRA__NOM) 49 0.9245283)
((MUSICA__NOM CLASSICO__ADJ) 43 0.9772727)
((COVER__NOM) 39 1.0)
((ALBUM__NOM) 39 0.8125)
((ORCHESTRA__NOM) 37 0.948718)
((PIANISTA__NOM) 33 1.0)
((GRUPPO__NOM ROCK__NOM) 31 0.9393939)
((MUSICA__NOM ITALIANO__ADJ) 28 0.8)
((ITALIANO__ADJ ROCK__NOM) 27 0.9642857)
((ASSOCIAZIONE__NOM MUSICALE__ADJ) 27 1.0)
((DATA__NOM CONCERTO__NOM) 25 1.0)
((LIVE__NOM) 25 0.8333333)
((SITO__ADJ GRUPPO__NOM) 25 0.6097561)
((COMPOSIZIONE__NOM) 23 0.7419355)
((BATTERIA__NOM) 22 0.9565217)
((GOSPEL__NOM) 22 1.0)
((CANTO__NOM) 22 0.6875)
((PIANOFORTE__NOM) 21 0.9130435)
((METAL__NPR) 21 0.9545454)
((CHITARRISTA__NOM) 20 1.0)
((FANS__NOM CLUB__NOM) 20 0.7142857)
((TESTO__NOM CANZONE__NOM) 20 0.8)
((BEATLES__NOM) 18 1.0)
```

While it is true that most of the features are still terms (which shouldn't surprise: in this context, some terms are specific enough), as it happened before to e. g. Mitchell [7], 'it is also true that the ambiguous terms are either absent or specified.

There is no feature composed of the single term `GRUPPO` anymore, while more specific and meaningful features substituted it: `(GRUPPO MUSICALE)` (musical group) and `(GRUPPO ROCK)` (rock band). In terms of meaning representation, the features of depth 1 give a good result.

When the features have structure of depth 2, this is part of the profile we obtain:

```
(((BAND__NOM)) 178 0.978022)
(((ROCK__NOM)) 171 0.9095744)
(((CANZONE__NOM)) 121 0.8705036)
(((DISCOGRAFIA__NOM)) 105 0.9813084)
(((JAZZ__NOM)) 98 0.9245283)
(((MUSICA__NOM) (MUSICA__NOM)) 76 0.7755102)
(((BRANO__NOM)) 75 0.8522727)
(((GRUPPO__NOM MUSICALE__ADJ)) 73 0.9864865)
(((SITO__NOM) (MUSICA__NOM)) 73 0.6759259)
(((MPT__NOM)) 71 0.8987342)
(((MUSICISTA__NOM)) 70 0.9090909)
(((REPERTORIO__NOM)) 67 0.8589743)
```

```
(((DISCO__NOM)) 57 0.721519)
(((CORO__NOM)) 56 0.9491525)
(((CANTANTE__NOM)) 54 0.84375)
(((CONCERTO__NOM) (GRUPPO__NOM)) 48 0.96)
(((BLUES__NOM)) 47 0.94)
(((SITO__NOM) (CANZONE__NOM)) 43 0.9148936)
(((ROCK__NOM) (SITO__NOM)) 40 0.8888889)
(((MUSICA__NOM CLASSICO__ADJ)) 40 0.9756098)
(((COMPOSITORE__NOM)) 40 0.8695652)
(((ALBUM__NOM)) 39 0.8125)
(((BAND__NOM) (SITO__NOM)) 39 0.975)
(((CHITARRA__NOM)) 38 0.9047619)
(((COVER__NOM)) 38 0.974359)
(((GRUPPO__NOM) (BAND__NOM)) 36 1.0)
(((CURRICULUM__NOM)) 36 0.4615385)
(((MUSICA__NOM) (ROCK__NOM)) 34 0.9444444)
....
....
(((CONCERTO__NOM) (FOTO__NOM)) 22 0.9565217)
....
....
(((TESTO__NOM) (MUSICA__NOM)) 18 0.8571429)
(((ASSOCIAZIONE__NOM CULTURALE__ADJ)
   (MUSICA__NOM)) 17 0.7727273)
(((FESTIVAL__NOM) (MUSICA__NOM)) 15 0.7894737)
```

This profile has about the same characteristics that were shown for the profile made of features of depth 1. While still many terms are left alone as unambiguous terms, specialization can bring a term inside the feature without it to necessary belonging to the same noun phrase. This makes the generalization of features more elastic, and thus information can be represented in a more concise way.

We can still consider the term `GRUPPO`. Once again, the term does not appear by itself in a feature, since it must have caused an ambiguity. While we still have the `(GRUPPO MUSICALE)` specification we pointed out in the case of feature of depth one, we see that the term can be specified differently: `((GRUPPO) (CONCERTO))`. By adding to the feature the term `CONCERTO` (concert), in a separate noun phrase, the meaning of the term narrows down, and can be connected to the category `MUSICA` with no more doubt. For this reason, its weight is fairly high (0.96). Even more clear is the case when `GRUPPO` was specified as `((GRUPPO) (BAND))`, which left no ambiguity to the system at all: the weight is 1.

While the description of the category provided by this last profile seems a good one, we must not forget that for as precise it can be, it could be incomplete. As we said before, more specific features make us run the risk to "miss" part of the document. To give an example, if, in the profile based on terms, the feature `MUSICA` was found with a relatively high weight in 531 examples for the category, its specification, while making the weight higher, could miss many documents. Specifications as `(MUSICA CLASSICA)` (classical music), while leaving fewer uncertainties about its meaning, is only found in 43 examples. This is the reason while we would want to still use an only vaguely ambiguous term such as `MUSICA` as well. By a post-generalization it will be possible to reinsert the feature `MUSICA` in the profiles of feature of depth 1 or 2. After a post-generalization step performed over the last profile we showed, and before any "cleaning" step, the profile looks as follows:

```
(((MUSICA__NOM)) 431 0.7019544)
(((SITO__NOM)) 354 0.1589582)
```

```
(((GRUPPO__NOM)) 256 0.4149109)
(((CONCERTO__NOM)) 180 0.8256881)
(((BAND__NOM)) 178 0.978022)
(((ROCK__NOM)) 171 0.9095744)
(((INFORMAZIONE__NOM)) 135 0.106383)
(((FOTO__NOM)) 135 0.2890792)
(((PAGINA__NOM)) 131 0.1884892)
(((CANZONE__NOM)) 121 0.8705036)
(((DISCOGRAFIA__NOM)) 105 0.9813084)
(((JAZZ__NOM)) 98 0.9245283)
(((NOTIZIA__NOM)) 93 0.1728625)
(((ASSOCIAZIONE__NOM)) 90 0.113924)
(((STORIA__NOM)) 89 0.1724806)
(((TESTO__NOM)) 88 0.5116279)
(((MUSICA__NOM) (MUSICA__NOM)) 76 0.7755102)
(((BRANO__NOM)) 75 0.8522727)
(((BIOGRAFIA__NOM)) 74 0.5736434)
(((MONDO__NOM)) 73 0.1054913)
(((SITO__NOM) (MUSICA__NOM)) 73 0.6759259)
(((GRUPPO__NOM MUSICALE__ADJ)) 73 0.9864865)
(((SITO__NOM) (GRUPPO__NOM)) 72 0.5413534)
(((MPT__NOM)) 71 0.8987342)
(((LINK__NOM)) 70 0.1670644)
(((MUSICISTA__NOM)) 70 0.9090909)
(((THE__NOM)) 68 0.3820225)
(((SITO__NOM UFFICIALE__ADJ)) 67 0.2617188)
(((REPERTORIO__NOM)) 67 0.8589743)
(((ANNO__NOM)) 66 0.1958457)
```

The term `MUSICA` has been reinserted into the profile, but along with it, many irrelevant terms have as well. Because the term `MUSICA` was only slightly ambiguous, however, its weight, as recalculated, is still relatively high: ~ 0.7. Other terms, such as `SITO` or `LINK`, which were clearly undesirable, have very low weights.

This allows a "cleaning" step, which would remove all the features whose weight is under a certain threshold. the threshold is set as to twice the percentage of the documents belonging to the category. Because the category `MUSICA` has 0.1214 (12.14%) of the examples in the whole collection, the threshold is set to about 0.25. This will cause the elimination of the following features that were added vie the post-generalization step:

```
(((SITO__NOM)) 354 0.1589582)
(((INFORMAZIONE__NOM)) 135 0.106383)
(((PAGINA__NOM)) 131 0.1884892)
(((NOTIZIA__NOM)) 93 0.1728625)
(((ASSOCIAZIONE__NOM)) 90 0.113924)
(((STORIA__NOM)) 89 0.1724806)
(((MONDO__NOM)) 73 0.1054913)
(((LINK__NOM)) 70 0.1670644)
(((ANNO__NOM)) 66 0.1958457)
```

All of them are irrelevant to the category, and their elimination does nothing but enhance the quality of the description. The constant to be multiplied to the percentage of the documents belonging to the category could be varied to force stronger or weaker reductions.

It is also possible to control which terms were "stopped" by the stop word removal technique we deployed. The stop list changes with time, so we will look at the list at the end of the training, when the most useless terms should have been identified. This is the top part of the stop list:

```
SITO__NOM
INFORMAZIONE__NOM
ASSOCIAZIONE__NOM
MONDO__NOM
RICERCA__NOM
GRUPPO__NOM
SERVIZIO__NOM
PAGINA__NOM
ITALIA__NPR
STORIA__NOM
ATTIVITA__NOM
NOTIZIA__NOM
```

As it is obvious, these are all terms that cannot be connected to the meaning of any category without uncertainty. The peculiarity of the domain, in which not many terms are used, makes this terms be few. It is encouraging, however, to see that stopped terms are of the kind that would not usually included in a stop list, and that the system recognized these terms correctly basing itself on ambiguity.

Other terms that are as ambiguous, such as `LINK` were not included as their frequency was lower, and thus caused fewer problems.

9.4 Classification analysis

The classification produced by the system is analyzed in two way:

- *qualitative* analysis: by manually going through the decisions taken by the classifier, we try and understand the reasons for its behavior. This was a priceless tool for development and improvement, and can give an idea of the functioning of the final system;
- *quantitative* analysis: classification is evaluated by standard evaluation measures.

9.4.1 Decision analysis

By reading the decisions taken by the classifiers and their motivation, we verify whether the system finds the proper features (concepts) inside the documents to be classified. This way, we also have a mean to judge whether its errors are caused by inconsistencies or opinability in the source classification.

In the following, we give some examples of decisions taken by the classifier upon some documents. The classification function used in these examples is comb-inf-pos.

```
Documento: (((ITALIA . NPR))
            ((ASSOCIAZIONE . NOM) (CULTURALE . ADJ))
            ((SCOPO . NOM))
            ((VALORIZZAZIONE . NOM) (PATRIMONIO . NOM)
             (CULTURALE . ADJ) (NAZIONALE . ADJ))
            ((PAESAGGIO . NOM))
            ((AMBIENTE . NOM))
            ((STORIA . NOM))
```

```
            ((ARTE . NOM))
            ((GRUPPO . NOM) (INTELLETTUALE . NOM))
            ((ARTISTA . NOM))
            ((PROFESSORE . NOM) (UNIVERSITARIO . ADJ)))

Categoria/e corretta/e: (ARTE_E_CULTURA)
ARTE_E_CULTURA  0.819 (0.812)
((ASSOCIAZIONE__NOM) (ARTE__NOM)): 0.7
((PAESAGGIO__NOM) (ARTISTA__NOM)): 1.0
((ITALIA__NPR)): 0.2
((STORIA__NOM)): 0.2
((SCOPO__NOM)): 0.2
```

This is an example of correct classification. The features found in the document for the category `ARTE_E_CULTURA` are five. The document is then classified relying on the occurrence in it of the features: `((ASSOCIAZIONE__NOM) (ARTE__NOM))` (art association) and `((PAESAGGIO__NOM) (ARTISTA__NOM))` (landscape artist), since they are the ones of highest weight. The second one is sufficient, by itself, to place the document in the category, since its weight is 1 (all documents examined containing the feature belong to the category).

The use of features made of just one term is still important for classification means:

```
Documento: (((ALBERGO . NOM) (GRAN . ADJ) (FANES . NOM))
            ((ALBERGO . NOM))
            ((SITUARE . VER-PPER) (CORVARA . NOM)
             (CUORE . NOM) (DOLOMITE . NOM))
            ((VACANZA . NOM) (IDEALE . ADJ) (INVERNALE . ADJ)))

Categoria/e corretta/e: (VIAGGI_E_TURISMO)
VIAGGI_E_TURISMO  0.998 (0.932)
((VACANZA__NOM)): 0.9
((ALBERGO__NOM)): 0.9
((CUORE__NOM)): 0.5
```

The two features `((VACANZA))` (holiday) and `((ALBERGO))` (hotel) are very representative of the category, and determine the assignment to it. The feature `((CUORE))` (heart) is not really related to the meaning of the category, and it is caused by the frequency of expressions such as "in the heart of the Alpes". Its weight is "so-so" (0.5) as it probably shares documents that belong to the category about medicine.

We will now examine a case in which classification was wrong.

```
Documento: (((GALLERIA . NOM) (FOTOGRAFICO . ADJ) (MAURIZIO . NPR)
             (PAPUCCI . NOM)   (MAURIZIO . NPR) (PAPUCCI . ADJ))
            ((GIORNALISTA . NOM))
            ((FOTOREPORTER . NOM) (QUOTIDIANO . NOM))
            ((NAZIONE . NOM))
            ((FOTOGRAFIA . NOM))
            ((BIANCO . ADJ) (NERO . ADJ))
            ((VIAGGIO . NOM) (LAVORO . NOM) (PARTO . NOM)
             (PIU . ADV) (POVERO . ADJ) (PIANETA . NOM)))

Categoria/e corretta/e: (ARTE_E_CULTURA)
INFORMAZIONE  0.929 (0.59)
((QUOTIDIANO__NOM)): 0.8
```

```
((VIAGGIO__NOM)): 0.1
((GIORNALISTA__NOM)): 0.6
Corretta non scelta ARTE_E_CULTURA   0.52 (0.812)
```

It is easy to identify the cause of the wrong classification. One of the assumptions made was not, in this case, valid. In particular, the use of commentary was misleading to the system, since by the description upon which the classifier takes his decision, the classification it gave could as well have been correct. The text from which the description shown above is extracted is:

La galleria fotografica di Maurizio Papucci

Maurizio Papucci, giornalista e fotoreporter del quotidiano "La Nazione" racconta attraverso le fotografie (in bianco e nero) i suoi viaggi di lavoro nelle parti più povere del pianeta.[18]

The "evidence" of the category ARTE_E_CULTURA in which the document is classified is only a minor part of the document, which could as well be classified before INFORMAZIONE as the system did, recognizing many press-related concepts.

It is interesting to verify that the mentioned document *is*, as a matter of fact, classified under INFORMAZIONE as well, before Virgilio. Because our spidering of the site was incomplete, however, such page was not retrieved, and we only saw the classification of the document before ARTE_E_CULTURA.

There are cases in which the classification is wrong because of subjective circumstances. The catalog might give "peculiar" interpretations of a meaning, over all the collection or on a specific document:

```
Documento: (((FANTASY . NOM) (LAND . NOM))
            ((SITO . NOM) (PSICOLOGIA . NOM))
            ((DISAGIO . NOM) (PSICHICO . ADJ))
            ((SPECIALISTA . NOM) (CLINICO . ADJ))
            ((CONSULENZA . NOM) (GRATUITO . ADJ)))

Categoria/e corretta/e: (UNIVERSITA_E_SCIENZE)
SALUTE_E_MEDICINA  0.933 (0.69)
((SPECIALISTA__NOM)): 0.7
((CONSULENZA__NOM GRATUITO__ADJ)): 0.8
Corretta non scelta UNIVERSITA_E_SCIENZE  0.638 (0.773)
```

The concepts of CONSULENZA GRATUITA or SPECIALISTA CLINICO hint at a category such as SALUTE_E_MEDICINA, more than UNIVERSITA_E_SCIENZE, as the document was classified under in Virgilio. The decision taken by the classifier is not objectively criticizeable in this case. The "error" could be seen as a different "interpretation".

It is possible that a document is not recognized at all, and is therefore assigned to no category.

```
Documento: (((RASSEGNA . NOM) (STRUMENTO . NOM)
             (INFORMATICO . ADJ))
            ((STUDIO . NOM) (ANTICHITA . NOM) (CLASSICO . ADJ))
            ((ELENCO . NOM) (MOLTO . ADV) (FORNIRE . VER-PPER)
             (COLLEGAMENTO . NOM))
            ((AMPIO . ADJ) (GAMMA . NOM) (STRUMENTO . NOM)
```

[18] "Maurizio Papucci's photographic gallery – Maurizio Papucci, journalist and photo-reporter of 'La Nazione' newspaper, tells his working travels in the poorest parts of the planet by his photographs (in black and white)".

```
             (INFORMATICO . ADJ))
            ((SPUNTO . NOM) (INTERESSANTE . ADJ) (RICERCA . NOM))
            ((INFORMAZIONE . NOM) (ECC . ADV)))

Categoria/e corretta/e: (UNIVERSITA_E_SCIENZE)
(NONE) ----> Non scelto:
UNIVERSITA_E_SCIENZE  0.427 ( 0.773)
((INFORMAZIONE__NOM)): 0.1 185 0.0 2.1
((RICERCA__NOM) (STUDIO__NOM)): 0.5 36 0.0 5.0
((STRUMENTO__NOM)): 0.1 10 0.0 4.9
Corretta non scelta UNIVERSITA_E_SCIENZE  0.427 ( 0.773)
```

The document does not surpass the threshold of any category.

In this cases, it might be worth to apply a "recover" technique as described in 8.6. Such a technique is especially worth in our case, when we know that each document is assigned to at least a category. In our experiments, we have seen that when the threshold was not surpassed, we could still give the correct classification in the majority of cases.

The matching function that gave the best results has been the one combining positive and negative evidence.[19] Because such a function takes into account the features found in the profiles of other categories, the decision module is different, for it, so as to show also the "negative" matches. This is an example of its behavior:

```
Documento: (((MAURIZIO . NPR) (MERLIN . NOM) (PORTFOLIO . NOM)
             (FOTOGRAFICO . ADJ))
            ((FOTO . NOM) (VIAGGIO . NOM))
            ((REPORTAGE . NOM) (TRADIZIONE . NOM) (VENETO . ADJ)
             (MAURIZIO . NPR)   (MERLIN . NOM)))

Categoria/e corretta/e: (ARTE_E_CULTURA)
ARTE_E_CULTURA  0.344 ( 0.205)
Categoria: ARTE_E_CULTURA
Pos-match-list
 ((REPORTAGE__NOM)) 0.6470588
 ((PORTFOLIO__NOM FOTOGRAFICO__ADJ)) 1.0
Neg-match-list
 ((TRADIZIONE__NOM)) 0.699115
 ((VIAGGIO__NOM)) 0.7385621
 ((FOTO__NOM)) 0.8501071
```

The feature in the first group ("pos-match-list") are the features found in the document that belong to the profile of the category. The ones in the group called ("neg-match-list") are the features in the document belonging to other categories, and that therefore can make it less likely that the document belong to the first category. In this case, `REPORTAGE` and `PORTFOLIO FOTOGRAFICO` are "symptoms" of the category `ARTE_E_CULTURA`, while the other features make a category like `VIAGGI_E_TURISMO` comes to mind more than anything else.

9.4.2 Evaluating classification

While the analysis of the decisions one by one has proven very effective and descriptive of the behavior of the system, it is not feasible to read through all the examples, and there's a need of a measure of how good classification is.

[19]It is the case to remind that by "negative evidence", here, we do not indicate the absence of a feature, but the presence of a "bad" feature instead.

Depth	# profile features	$\mathcal{F}_1$ microaveraged
0	8768	0.6983478
1	10080	0.6684116
2	2913	0.7008016

Figure 18: **table with the best results achieved by the different instantiations.**

To evaluate the results of classification, we used the module that sorts the documents, and examined the percentage of correctly retrieved documents amongst the number that originally were assigned to the category. This measure is equivalent to $\mathcal{F}_1$ microaveraged when the system is set to reach the *break-even point* (the point where recall equals precision). A perfect classification would achieve 100%, and no relevant document would be left out.

The best results obtained for each one of the three instantiations, with different parameters setting, is shown on figure 18.

The conditions with which such results were obtained are:

- feature of depth 0: features which subsumed fewer than 3 examples where eliminated from the profiles; cosine measure was deployed as matching function, weights were assigned by inf-pos;

- feature of depth 1: features subsuming fewer than 2 examples were eliminated from the profiles. The matching function used was the one using negative evidence.

In figure 19 we show the results obtained with the various depth of features. Documents are ordered as stated above. We can see how performance changes form category to category. The set of documents ordered is obviously the same in all three cases.

We must interpret the case of features of depth 0 as a case analogous to the classic techniques. While such techniques were deployed, and discarded when it was obvious that no improvement was obtained, we must remember here that the system includes flawlessly such techniques, and that their performance is therefore still achievable.

From the figure, we see how the more structured features perform better when more examples are available. We must take into account that examples are not the documents, so the instantiation with more complicated features had fewer examples (but of better quality) than the others.

In figure 20 the learning curve of the system for various matching function is shown. The curves are relative to the case of features of depth 2. Because the functions have approximately the same slope, only translated, we believe that the best results are still to be obtained, and that would happen when a better classification scheme is developed. From the graphs, however, it is evident how the elimination of dependent features improved performance: while comb-inf-pos is the is the function we described elsewhere, scomb-inf-pos is a function with the same structure, but which does not evaluate any dependence amongst features, and simply considers them all independent.

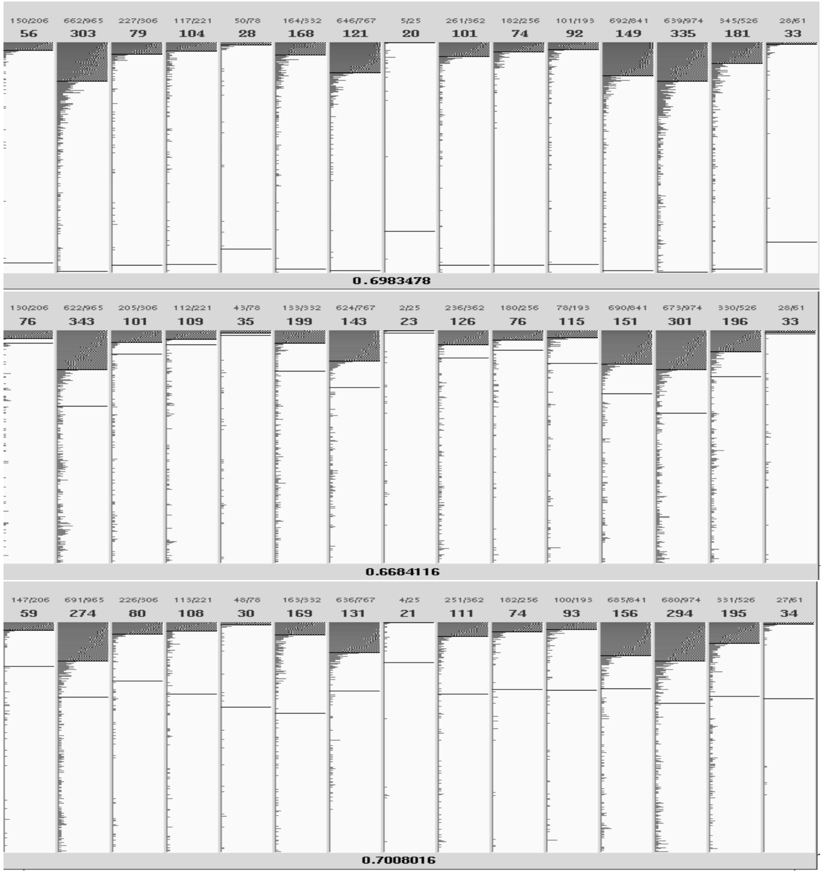

Figure 19: **documents ordered according to the profiles with features of depth 0, 1 and 2 (from top to bottom).**

10 Conclusions

10.1 The goal and the accomplished work

In this work we approached the problem of building conceptual descriptions of categories from pre-categorized documents. We have pointed out as in literature such a goal is uncommon, as much research in the same domain engages automatic classification directly. We have also shown how the goal we wanted to achieve could constitute a first step in an improvement in classification performance which seems infeasible by statistical methods.

We defined a general model which, thanks to its capacity to represent general textual features, captures most of the most common methods oriented to the classification, letting, at the same time, some new approach emerge, which we deployed to obtain our main goal.

We defined a class of algorithms which can be implemented by assigning some parameters in order to to obtain descriptions of categories based on textual features, by feeding the algorithm with a set of pre-categorized examples.

Features are extracted with a *generalization* process, which finds commonalities amongst example of the same category, and a *specialization* process, which detects and fix ambiguous or irrelevant features.

The general definition we gave of the structure of a features allows, from the side of the

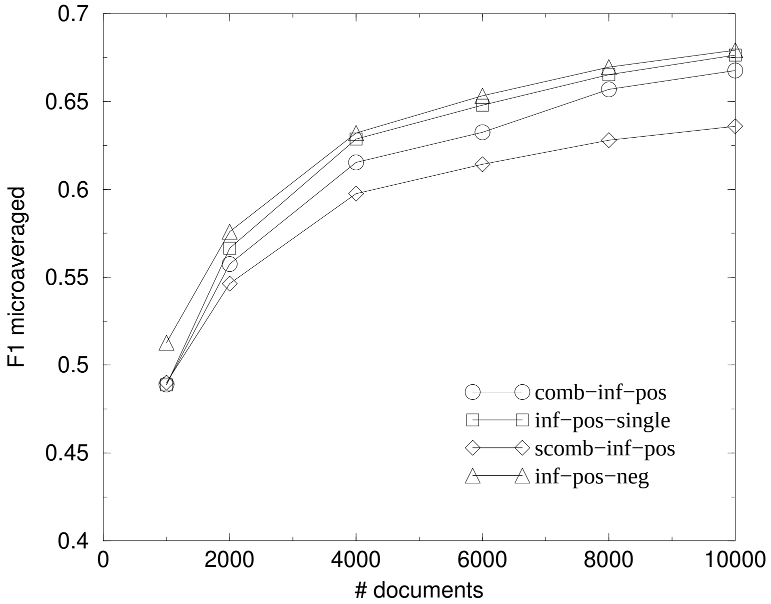

Figure 20: **learning curve obtained with different matching functions. The function "scomb-inf-pos", which does not take the dependence of features into account, performs worse than all others. The curve is for features of depth 2.**

algorithm, to treat terms or more structured features in the same way. We have used this first case as analogy with the terms-based classifiers, while the latter were deployed with the assumption that they represent *concepts*, and were therefore used as a mean to accomplish our goal. The generality of the model goes as far as allowing an instantiation that actually makes of the system a word stemmer from examples.

We gave three different instantiations of the proposed algorithm: one based on simple terms, so as to have an indirect comparison with standard classifiers and to verify the elasticity of the system, one based on noun phrases and one based on lists of noun phrases. We represented documents by *commentaries* instead of by content, following indications of some of the most recent works on classification.

We studied consolidation techniques for the profiles, in order to improve the quality of the descriptions. Also, techniques for boosting the training phase were adopted, and specifically defined for our special case.

The features obtained were weighted according to some of the most common measures such as Rocchio or TF-IDF as well as to some measure defined *ad hoc*.

We have considered the opportunity to use the generated profiles for classification. We have deployed usual techniques as well as "original" ones, oriented to a better exploitation of the knowledge in the profiles.

We gave an implementation of the system with which we verified the behavior of the system and experiment on a real ground. The implementation allowed to change some of the model's parameters without changing the code, which was useful to verify the theoretical intuition. Several control modules to monitor the training phase were developed, and constituted priceless support during development. As well as the training phase, the system is complete with a thresholding system, a classifier and an evaluator.

The implementation of the system has been run on a collection of commentaries obtained from the Web, which gave us indications upon its behavior.

10.2 Evaluation

This work was a pure research work. The original problem has been studied, looking for a solution which would go beyond the statistical approaches, whose limits seem obvious. Thanks to ideas taken from other works, it was possible to define a model which, approaching the problem from a different viewpoint, could generalize previous models.

The model if reasonable and satisfactory. The algorithm class it allows the definition of, instantiated and implemented, performed as expected, especially in the case for which we had more expectations: the case of features of depth 2.

The profiles are very interesting: considering the strong initial assumptions (no verbs, the order of words does not matter), results are very encouraging. If we take the difficulty of the training set chosen into account, we observe how the most typical concepts for each category tended to emerge in the profiles, without the use of any information about their frequency if not in post-processing phases. The spontaneous choice of concepts such as "musical group" to represent the category labeled as "Music" without the use of any external information on how to form concepts if not the training examples is a clear example of the achievements of the algorithm. Also, we are further encouraged by the chance to use the many textual information we discarded, which could make performance even better.

Tests conducted on classification have shown results that are not unlike the ones usually obtained in literature. On one side, the positive results obtained by statistical methods are known, while on the other, the bad results obtained by using more structured features discourages further study on that direction. The system we proposed obtained similar results in both cases, which is to be considered a success and an encouraging results toward an approach to classification that goes beyond statistics.

10.3 Considerations and interpretations

It is possible to make some considerations upon the results obtained which, since they are not proven, will only remain as subjective interpretations. In information retrieval, it is acknowledged that using anything but single terms is an effort that is likely to be wasted, since the best results, no matter the effort, are obtained by simple statistical term-based methods. Contrarily to this widely accepted assumption, however, there are many examples of research on other methods. Apparently, many people feel that there is something behind the wall, and that there could be a way to deploy the more information available in features composed of more than one term. So far, however, results have been disappointing.

We think that, while we still did not obtain sensational classification results, an indication could be given by the present work. In particular, the case of features of depth 1 is emblematic. With such a depth, results dropped down significantly from the ones obtained by single terms, just as experimented other times by researchers. By insisting, that is, by considering features of depth 2, performance was back. While we cannot be as simplicistic as saying that a depth of 3 would therefore achieve even better solution, we will state anyway that it is possible that the lower performance of depth 1 is only a local minimum. Maybe, the assumptions made at that depth are too strong, but a deeper analysis could get over those problems, and make classification results better.

We believe that study oriented to linguistic analysis and computation could bring great improvement, even though we know that a full text understanding is not very likely. We did very basic and naive language analysis, still we believe that our results owe a lot to such analysis. Statistical methods will probably always be the most economical and fast systems; they will always guarantee good accuracy at very low resource costs. We think, however, that very high-end (or critical) applications would rather deploy a more complex system if it grants a better performance. We must say, however, that as resources become less of an issue, it is possible that complications of statistical models themselves (e. g. releasing the independence assumption) could achieve very good results as well.

Further considerations could be made about concept representation. The use of lists of noun phrases still brought to the formation of lists of just one element: in most cases the single phrase was enough to express a non ambiguous (by our definition) concept. This feeling is also caused by the results obtained by the function that privileges features composed of just one noun phrase. The use of lists of noun phrases seems appropriate for a deeper analysis.

10.4 Possible applications

Amongst the possible applications of the system there is, obviously, automatic categorization. If the instantiation given to "emulate" other systems is a degeneration of the model, other cases proved to be interesting on their own, and we think that ad hoc classification techniques to deploy the peculiarities of such approaches could improve results.

Classification is not, however, the only application of the system. As textual features are found, it could be used to integrate or support linguistic tools, by expressing relationships amongst concepts or textual features, or by marking the n-grams more related to a concept.

The extracted features could then be used by systems for text understanding: they give a knowledge of the domain that could be use to extract the meaning of a text.

10.5 Future work

As our work was quite spread, many of the aspects described here could be studied further and improved, in a short or long term plan. As for a short term plan, porting to different languages and making the system availability as an embeddable component is planned. Other short term plans could include the improvement of the tagger and of the noun phrase extraction.

A deeper natural language analysis should be integrated in the system, and would require more study. We think that a complete analysis would be over-killing, but improvements could be done with smaller efforts.

A system to eliminate bad examples in the case of features with less structure than the documents is highly desirable, as we feel it would reduce the profile size in many cases. This could be achieved with a technique not unlike the one we used for stop word removal.

The many parameters of the system need more accurate study, to find optimal value for all of them. If an optimal value for a parameter is found independently from all others, it might as well be assigned permanently, and thus made a constant.

The use of external information must be studied and allowed.

The very ambitious planned work, however, is the one to extract not only plain concepts, but real meaning, by finding relationships between concepts. A representation of the meaning of a document in some structured way would allow e. g. some form of question answering, or the deploy of techniques of data mining to obtain structured informations from a large database of textual documents.

References

[1] *Altavista*, http://www.altavista.com

[2] *Arianna*, http://arianna.it

[3] Giuseppe Attardi, Antonio Gullì, Fabrizio Sebastiani, *Theseus: Categorization by Context*, 8th World Wide Web Conference, Toronto, Canada, 1999

[4] Gustzi Bartfai, *An Adaptive Resonance Theory-Based Neural Network Capable of Learning via Representation Redescription*, In Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN'98), pp. 1137-1142.

[5] *Excite*, http://www.excite.com

[6] Eibe Frank, Gordon W. Paynter, Ian H. Witten, *Domain-Specific Keyphrase Extraction*, Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, 1999

[7] Johannes Fürnkranz, Tom Mitchell, Ellen Riloff, *A Case Study in Using Linguistic Phrases for Text Categorization on the WWW*, Working Notes of the AAAI/ICML Workshop on Learning for Text Categorization, 1998.

[8] Paul Graham, *ANSI Common Lisp*, Englewood Cliffs, Prentice Hall, 1996

[9] Lucien Heins, Daniel Tauritz, *Adaptive Resonance Theory (ART): An Introduction*, IR-95-35, Department of Computer Science, Leiden Univeristy, 1995

[10] David J. Ittner, David D. Lewis, David D. Ahn, *Text Categorization of Low Quality Images*, Symp. on Document Analysis and Information Retrieval, April, 1995, pp. 301-315.

[11] Thorsten Joachims, *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*, Proceedings of International Conference on Machine Learning (ICML), 1997.

[12] Pat Langley, Wayne Iba, Kevin Thompson, *An Analysis of Bayesian Classifier*, Proceedings of the 10^{th} National Conference on Artificial Intelligence, San Jose, 1992

[13] David D. Lewis, *Representation Quality in Text Classification: An Introduction and Experiment*, Proceedings of Speech and Natural Language Workshop, Defense Advanced Research Projects Agency, Morgan Kaufmann, June 1990, pp. 288-295.

[14] David D. Lewis, *Term Clustering of Syntactic Phrases*, Proceedings of the 13^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-90), Brussels, 1990

[15] David D. Lewis, *Evaluating Text Categorization*, Proceedings of the Speech and Natural Language Workshop, Asilomar, Feb. 1991, Morgan Kauffman, San Mateo, CA, pp. 312-318

[16] David D. Lewis, *Learning in Intelligent Information retrieval*, Machine Learning: Proceedings of the 8^{th} International Workshop (ML91), San Mateo, CA: Morgan Kaufmann, 1991

[17] David D. Lewis, *An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task*, In 15^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 37-50, 1992.

[18] David D. Lewis, Robert E. Shapire, James P. Callan, Ron Papka, *Training Algorithms for Linear Text Classifiers*, SIGIR-96: Proceedings of 19^{th} Annual International ACM SIGIR Conference on Research and Developmente in Information Retrieval, Zurich, CH, pp. 298-306

[19] David D. Lewis, *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval*, European Conference on Machine Learning, 1998

[20] Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997

[21] John K. Ousterhout, *Tcl and the Tk toolkit*, Reading, Addison Wesley Publ., 1994

[22] Miguel E. Ruiz, Padmini Srinivasan, *Automatic Text Categorization Using Neural Networks*, Proceedings of the ASIS SIG/CR Classification Research Workshop, ed. Efthimis Efthimiadis. Information Today, Medford: New Jersey, 1998

[23] Stuart Russel, Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, US, 1995

[24] Gerard Salton, *A Blueprint for Automatic Indexing*, SIGIR Forum, 1981

[25] Tommaso Schiavinotto, Filippo Tanganelli, Alessandro Tommasi, *D.E.U.S.: Eleusis Unpredictable Scientist*, Proceedings of the Associazione Italiana per l'Intelligenza Artificiale (AI*IA99), Bologna, 1999

[26] Helmut Schmid, *Probabilistic Part-of-Speech Tagging Using Decision Trees*, Proceedings of International Conference on New Methods in Language Processing, September 1994

[27] Hinrich Schütze, *Distributional Part-of-Speech Tagging*, Proceedings of the 7^{th} Conference of the European Chapter of the Association for Computational Linguistics, pp. 141-148, Dublin, Ireland, 1995

[28] Luc Steels, *Perceptually grounded meaning creation*, Tokoro, M. (ed.) Proceedings of ICMAS-96, Kyoto, JP, 1996, AAAI Press.

[29] Luc Steels, *Constructing and Sharing Perceptual Distinctions*, van Someren, M. and G. Widmer (eds.), Proceedings of the European Conference on Machine Learning, Berlin, 1997.

[30] *Structured Information Management: Processing and Retrieval*, http://www.lingsoft.fi/doc/nptool/intro/previous.html#simpr

[31] *Virgilio*, http://www.virgilio.it

[32] *Yahoo!*, http://www.yahoo.it

[33] Yiming Yang, Tom Pierce, Jaimie Carbonell, *A Study on Retrospective and On-Line Event Detection*, ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98), 1998

[34] Yiming Yang, *An Evaluation of Statistical Approaches to Text Categorization*, Information Retrieval Journal, May 1999